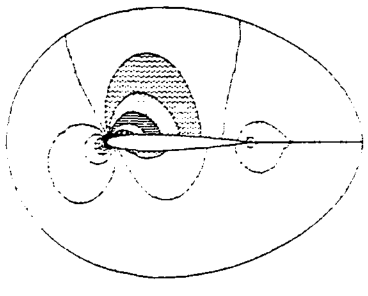
FINAL TECHNICAL REPORT

COMPUTATIONAL METHODS FOR COMPLEX FLOWFIELDS

Earll M. Murman
Judson R. Baron

Principal Investigators

AFOSR Grant 87-0218



COMPUTATIONAL FLUID DYNAMICS LABORATORY
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

2

FINAL TECHNICAL REPORT

COMPUTATIONAL METHODS FOR COMPLEX FLOWFIELDS

Earll M. Murman
Judson R. Baron

Principal Investigators

AFOSR Grant 87-0218

MIT OSP 99145, 70518

July 5, 1989

Computational Fluid Dynamics Laboratory
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR.TR. 89-1072
6a. NAME OF PERFORMING ORGANIZATION Dept. of Aero and Astro Mass. Institute of Technology; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION USAF Office of Scientific Research
6c. ADDRESS (City, State, and ZIP Code) Cambridge, MA 02139	7b. ADDRESS (City, State, and ZIP Code) Bolling AFB BK1 410 Washington DC 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION USAF Office of Scientific Research; 8b. OFFICE SYMBOL (If applicable) NA	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER AFOSR-87-0218
8c. ADDRESS (City, State, and ZIP Code) Bolling AFB BK1 410 Washington DC 20332	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.
61102F	2307	A1	

11. TITLE (Include Security Classification)
Final Technical Report - Computational Methods for Complex Flowfields - UNCLASSIFIED

12. PERSONAL AUTHOR(S)
Murman, Earll M.; Baron, Judson R.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final Technical Report	FROM 6/1/87 TO 5/31/89	July 5, 1989	286

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
	20.04	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Euler equations; Embedded grids; Navier-Stokes equations; Adaptive grids; Finite element methods; Computational fluid dynamics

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The development of solution algorithms for complex flowfields has been the objective of this research. Embedded subdomains were used to resolve relevant physical processes in a global flow around aerodynamic bodies. Adaptive approaches were studied and developed for the two- and three-dimensional Euler equations and two-dimensional Navier Stokes equations using finite volume and finite element methods. A new approach is reported for combining expert system approaches with adaptive procedural algorithms into a totally integrated methodology. Recent results on formulation of outflow boundary conditions for the Navier-Stokes equations and compact high-order schemes for the Euler equations are also presented. Additional tasks included studying the performance of CFD algorithms on several parallel processors, a short study on turbulent spot measurements, and the prediction of dispersive errors in numerical solution of the Euler equations.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] [✓ SAME AS RPT.] ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Leonidas Sakell	22b. TELEPHONE (Include Area Code) 202-767-4935; 22c. OFFICE SYMBOL NA

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

89 8 10 078 UNCLASSIFIED

TABLE OF CONTENTS

INTRODUCTION

During a two year grant period several research tasks have been completed in a continuation of the long range objectives related to the development of numerical methods for complex flow fields. The major emphasis has been the background concept of algorithms which recognize the extent to which the procedure matches the physical description being evaluated. This has involved extensions of our adaptive techniques to include finite element and Navier-Stokes components, as well as peripheral study of analytical and machine architecture possibilities for achieving accelerated convergence.

The physical impetus is the nature of fluid fields such that individual events often occur in relative isolation and therefore with a scale unique to the local physical feature. However, overlaps and interactions between features do occur and are of some importance. The numerical stimulus is the advantage that results from avoiding procedures carried out globally when a specific aspect of the discrete methodology is present solely to meet a local need. The research effort has focused on linking the two drives by way of unstructured grid and descriptive equation modifications that are introduced as the numerical integration is carried out and only where required in a space/time domain.

In a broader sense this amounts to achieving robust integration while ensuring computational efficiencies in both storage and time, and thus attention has also been given to boundary condition and parallel processing questions which are not limited to an adaptive approach.

Specific efforts have been devoted to embedded adaptation using finite element methods for both two and three dimensional steady state Euler systems [6, 11, 13], as well as finite difference methods for adaptive Navier-Stokes, steady and unsteady, two-dimensional fields [2, 12, 14-19]. Each has been demonstrated to be effective in terms of completed flow fields and their comparative efficiencies when measured against equivalent accuracies and resolution obtained with non-adaptive procedures. Non-reflecting outflow boundary conditions have also been given consideration for time consistent schemes for viscous flow, and evaluations have been completed to map out the behavior of various CFD algorithms on a number of parallel processor architectures [10, 20].

EMBEDDED ADAPTIVE ALGORITHMS

Adaptive Finite Element Methods

Work was completed on the development of adaptive finite element method (FEM) algorithms for 2 and 3 dimensional steady state Euler equations. The technical results are completely summarized in the PhD thesis of Richard Shapiro [11], as well as three papers at technical meetings [6, 9, 13]. A paper delivered at the January 1988 AIAA Aerospace Sciences meeting in Reno [6] and attached as Appendix 4 presents two-dimensional results for different FEM formulations. A second paper [13] presented at the January 1989 AIAA Aerospace Sciences meeting in Reno (Appendix 10) gave three-dimensional and new higher order (biquadratic) two-dimensional FEM results. The investigation into finite element algorithms led to an unanticipated inquiry into the source of dispersion errors for discrete Euler algorithms. These results were presented at the Second International Conference on Hyperbolic Problems; Theory, Numerical Methods and Applications, held in Aachen during March 1988 [9], and the paper is attached as Appendix 6. In the following paragraphs, the highlights of all these findings are discussed. The interested reader is referred to Shapiro's thesis or the papers for more details.

The major effort on developing adaptive finite element methods for the steady state Euler equations was spent on two-dimensional geometries in order to reduce computational expense and complexity. The general approach was to use FEM discretization in space, Runge-Kutta integration in time with local time stepping, characteristic boundary conditions on open boundaries, and added second and fourth order dissipation terms for suppressing background oscillations and capturing shock waves. Quadrilateral rather than triangular elements were used since they require less computer memory and are easier to generate. In theory, the quadrilateral elements are more accurate for a given shape function, but in practice this effect may not be noticeable. Triangular elements can be used where needed to fit body geometries.

The various topics which have been addressed for 2-D geometries included:

- For bilinear shape functions, different choices of test functions were analyzed and tried. The best choice was a constant test function which gives a scheme equivalent to a cell vertex finite volume method.
- The idea of using a special element when the mesh is exactly Cartesian was investigated. However, the saving in the overall operation count was not big enough to justify the additional coding requirements, and the idea was abandoned.
- Biquadratic shape functions were investigated for nonadaptive meshes. The results for the two-dimensional test problems proved to be very encouraging with significant saving in computational time for equivalent accuracy.
- Two different smoothing formulations were tried, a "low" order and a "high" order accurate version. It was found that the overall accuracy of the different FEM discretizations is more effected by the accuracy of the smoothing than the accuracy of the unverlying discretization method for the convective terms.
- Numerous calculations were done for scramjet inlet geometries and a bump in a channel. The adaptive FEM algorithm proved to be robust and capable of capturing shocks and slip surfaces.

Based upon the experiences of the 2-D studies, 3-D calculations were undertaken using the cell vertex FEM method. Calculations to verify the 3-D code were done for a double-wedge compression surface. Both non-adaptive and adaptive calculations were performed, but the latter should be considered as a first attempt. Further refinement needs to be done on adaption criteria for 3-D flows. To our knowledge, this represents the first adaptive 3-D hexahedral FEM calculations for the Euler equations. The final 3-D calculation was for a scramjet inlet. The basic features of the flow were resolved, but the lack of suitable computer capacity did not permit enough cells to be used. These results are reported in [11] and [13]. Calculations were initiated for a delta wing with the aim to model the leading edge vortex. However, results were not obtained by the conclusion of the grant. The effort will continue in a subsequent grant.

Lastly, in the course of the development of the different FEM algorithms, dispersion errors were evident for flows involving weak shock waves and expansion fans. This led to a new analysis of dispersion phenomena using spatial group velocity methods. The analysis correctly predicts whether dispersion phenomena ("wiggles") will appear ahead of or behind the disturbance and also predicts the wavelength of the oscillations. Different spatial discretizations, mesh aspect ratios, and local Mach number all influence these effects in a predictable way described by the theory. The analysis and results are given in [11] and [9].

Adaptive Navier-Stokes Methods

Our original embedded adaptation scheme (3) was limited to Euler modeling and therefore the capture of features which represented either large disturbance regions (e.g. the near field at an airfoil leading edge) or sharp discontinuities (e.g. shocks and slip surfaces representative of wakes). Fundamental aspects of the algorithm were formulated there to provide a proper data structure for the inherent unstructured grid that results for arbitrary features, as well as the modifications that might be required for internal interfaces created by the embedded grid scales.

The extension of the concept to allow Navier-Stokes modeling introduced a number of additional algorithm settings. These included:

- new discreteness modeling of the stencil for the second derivative viscous terms, to ensure the individual integration of an unstructured cell array, independent of each cell's neighbors for computational purposes
- allowance for multiple detection parameters, such as density differences for discontinuities and stresses for viscous diffusion regions, and their combined governance of the embedding needs within interaction regions.
- a directional embedding procedure in recognition of the frequent presence of elongated features such as shock fronts and boundary layers
- special implementation of turbulence models for consistency with the appearance of unstructured gridding and interfaces in regions near a boundary and/or within the core of a trailing viscous wake
- alternate choices of appropriate orders of conservation and accuracy for interfaces within interior and exterior regions of embedding with respect to a given kind of feature
- combined use of redistribution and embedding in establishing the initial stages of adapted regions with very highly concentrated fine gridding in large Reynolds number boundary layers
- a spatial variation of time steps corresponding to the spatial adaptation, to alleviate the stiffness associated with time accurate computations

A number of fields were calculated to demonstrate the algorithm's utility, including single element airfoils at transonic speeds ($M = 0.75$, Reynolds number = 3.8×10^6) and two element airfoils with different flap settings at low speed ($M = 0.19$, Reynolds number = 2.5×10^6). Detail with respect to separated regions, the presence and behavior of separation bubbles, gap flow between airfoil components, and shocks were captured with good fidelity using two to four adaptation levels.

Different aspects of the work have been reported at a number of forums. The initial Navier-Stokes effort was presented in Honolulu (2) and unsteady and turbulent modifications first appeared at an International Conference in Williamsburg and since then in the published proceedings (12). Two chapters of a book on viscous computations were prepared on invitation (16, 17) and some essential elements of the approach were included at an International Conference in Huntsville (14). A very detailed description of the algorithm, development, validations and applications is the subject of a doctoral thesis by Kallinderis (15), and final summaries have been prepared for submission to the AIAA Journal (18) and the next Reno Aerospace Sciences Meeting in January 1990 (19).

OUTFLOW BOUNDARY CONDITIONS AND COMPACT HIGH ORDER SCHEMES FOR NAVIER-STOKES EQUATIONS

Non-reflecting Downstream Boundary Condition for the Wake Problem

As indicated previously[1], the value of Ω, the dimensionless decay rate, and of ω_1, the dimensionless group velocity, for long wave perturbations was to be computed for a range of the parameter M, ε and K. This was done in 1988[2]. The main result is that for subsonic flow with high Reynolds number Ω is practically a constant, $\Omega \simeq \pi^2/4$, for all practical configurations ($.7 \leq K \leq .95$). The group velocity ω_1, for $M_\infty \leq .3$ and $K > .7$ is practically linear in K with the slope depending on the Mach number. For laminar flows, even for numerical boundaries relatively near the trailing edge of an airfoil - say about 5 chords away - the drag parameter K is near 1, namely $K > .91$. For this range of k, ω_1 is independent of M and is given approximately by

$$\omega_1 = 1.15 - 2.16K \quad (1 > K \geq .9)$$

The above values for Ω and ω_1 are appropriate for using in the non-reflecting boundary conditions when the algorithm used is time consistent. When, for the purpose of accelerating convergence to steady-state, one uses time-inconsistent schemes (such as by employing local time-stepping), one has to find a new set of Ω and ω_1. This requires further investigation.

[1] Final Technical Report, AFOSR Grant 82-0136, July 31, 1987.

[2] J.S.Danowitz, "A non-reflecting boundary condition for the compressible Navier-Stokes equations for two-dimensional wake flows," M.Sc. thesis, Tel-Aviv University, June 1988.

Compact High Order Schemes for the Euler Equations

It can be shown (9,11) that numerical approximations to the linearized Euler equations of gas dynamics give rise to dispersive errors which in the 2-D supersonic case depend on a similarity parameter $\kappa = (\Delta y/\Delta x)\sqrt{M_\infty^2 - 1}$ (under the assumption $v << u$ everywhere, where u and v are the x and y components of the velocity vector). The difference between the dispersion relations of any numerical algorithm and that of the original partial differential equations can be plotted as curves in the Fourier plane with κ as a parameter.

In particular the results in (9,11) indicate that for central-difference schemes, the dispersive errors are contributed mostly by the third power of the errors in the Fourier variables θ and ϕ. It is, therefore, natural to think of fourth order spatially accurate algorithms as having better dispersive properties. By utilizing the structure of the Euler equations one can obtain, on a Cartesian grid, a fourth order approximation which instead of using a 5×5 stencil (and $5 \times 5 \times 5$ in 3-D) relies on a compact support of 3×3 (and $3 \times 3 \times 3$ in 3-D). The advantages of the combination of fourth order accuracy together with compact support are quite obvious in terms of total computer work and memory.

In (7) we construct an implicit approximate factorization (AF) algorithm and a 4-step Runge-Kutta scheme which have the above properties of compactness and 4th order accuracy. In particular the following points should be noted:

- The 2-D AF scheme is unconditionally stable, has a 3×3 stencil and at steady-state has a fourth order spatial accuracy. The temporal evolution is time accurate either to 1st or 2nd order through choice of parameter.
- In 3-D the AF scheme has almost the same characteristics as in 2-D except that it is now only conditionally stable, with the stability condition (the CFL number) being dependent on the "cell aspect ratios" $\Delta y/\Delta x$ and $\Delta z/\Delta x$. The stencil is still compact and fourth order accuracy at steady-state is maintained.
- In the Runge-Kutta case it is shown how the Jameson-Schmidt-Turkel algorithms[1] may be easily modified to our form which has, in addition to the higher accuracy, markedly enhanced stability limits.
- Numerical experiments on a 2-D shock reflection problem (using the AF scheme) show the expected improvement over lower order schemes, not only in accuracy (measured by the L_2 error) but also in the dispersion.

[1] A.Jameson, W.Schmidt and E.Turkel, "Numerical solutions of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes," AIAA paper 81-1259, June 1981.

PERFORMANCE OF CFD ALGORITHMS ON PARALLEL PROCESSORS

The objective of this task is to explore how representative CFD algorithms map onto different parallel processor architectures. A careful analysis was made of an existing 3-D Euler solver using the Jameson cell-centere/ finite volume method in order to profile the typical computational loops in the program. The loops were characterized by the ratio of floating point operations to memory references and the span of mesh points (memory locations) required by the operation. Three characteristic loop structures were identified.

1. The first is represented by the calculation of the pressure given the values of the state vector. This has twice as many operations as memory references and involves data only at a single point in the grid (i,j,k).
2. The second is represented by calculating the surface flux integrals for a cell given the flux vectors at cell centers. This operation has about the same number of floating point operations as memory references and involves the nearest neighbors of a mesh point (i,j,k and $i \pm 1$, $j \pm 1$, $k \pm 1$).
3. The third loop is represented by the tridiagonal matrix solver and requires about one floating point operation for every two memory reference. It requires accessing a line of data through the mesh in each of three directions.

Although the analysis was done for an explicit Euler solver with residual smoothing, the loop structures are typical of other CFD codes using structured grids such as ARC3D.

In the first application, three smaller loops representing these computational tasks were written on two distinctly different architecture machines. The ALLIANT FX/8 has a shared memory and up to 8 processors while the INTEL iPSC has a distributed memory with up to 32 nodes (for the one we used). The loops are analyzed and reported in detail in [5, 8]. A paper summarizing the entire study was delivered at the 3rd International Conference on Supercomputing in Boston during May 1988 [10]. A copy is attached as Appendix 7.

In a second investigation [20] a 2-D version of the Jameson algorithm was implemented on a data-flow machine simulator using a new programming language called Id World. Whereas the ALLIANT and INTEL machines execute in parallel at the loop or subroutine level, a data flow machine executes in parallel at the elementary instruction level. The investigation revealed that CFD algorithms are suitable for data-flow architectures and such achieve much greater parallelism than conventional machines now in production.

TURBULENT SPOTS IN LAMINAR FLOW

A new set of measurements has been carried out to describe the transition process of a laminar flow over a flat plate as it evolves from a small amplitude wave-packet which has been generated by a controlled short duration air pulse, to the formation of a turbulent spot. The experiment was completed under the same conditions as an earlier first set of data with the exception that the initial amplitude level of the wave-packet was increased. The measurements are more dense along the downstream direction (X) and include a Y mapping of the flow field which was absent previously. Both streamwise and spanwise velocity components (U and W respectively) were measured with a V-shaped hot-wire anemometer while the transverse velocity component (V) was obtained by using an X hot-wire anemometer. A theoretical model which includes the flow divergent effect and a nonlinear analysis to capture the subharmonic stage is now in preparation and will be reported on at a later date. A summary of the results appears in Appendix 9.

CUMULATIVE LIST OF PUBLICATIONS

(1) Dannenhoffer, J.F. and Baron, J.R. "A Hybrid Expert System for Complex CFD Problems." AIAA 87-1111, Honolulu, June 1987.

(2) Kallinderis, J.G. and Baron, J.R. "Adaptation Methods for a New Navier-Stokes Algorithm." AIAA 87-1167, Honolulu, June 1987.

(3) Dannenhoffer, J.F. III. "Grid Adaptation for Complex Two-Dimensional Transonic Flows." MIT ScD Thesis, Aug. 1987. Also MIT report CFDL-TR-87-10.

(4) Loyd, B., Murman, E.M. and Abarbanel, S.S. "A Semi-Implicit Scheme for the Navier-Stokes Equations." Proceedings of 7th GAMM Conference on Numerical Methods in Fluid Mechanics, Sept. 1987.

(5) Modiano, D. "Performance of a Common CFD Loop on Two Parallel Processors." MIT report CFDL-TR-87-11, Sept. 1987.

(6) Shapiro, R.A. and Murman, E.M. "Adaptive Finite Element Methods for the Euler Equations." AIAA paper 88-0034, Jan. 1988.

(7) Abarbanel, S. and Kumar, A. "Compact High Order Schemes for the Euler Equations." ICASE Report No. 88-13, Feb. 1988. To appear in Journal of Scientific Computations.

(8) Haimes, R., Giles, M. and Murman, E.M.M. "Performance of Implicit Residual Smoothing on Multiprocessor Machines." MIT report CFDL-TR-88-2, Feb. 1988.

(9) Shapiro, R.A. "Prediction of Dispersion Errors in Numerical Solution of the Euler Equations." Proceedings of 2nd International Conference on Hyperbolic Problems, Aachen, Germany, March 1988. Also MIT report CFDL-TR-88-4.

(10) Murman, E.M., Modiano, D., Haimes, R. and Giles, M. "Performance of Several CFD Loops on Parallel Processors." Proceedings of 3rd International Conference on Supercomputing, Boston, MA, May 1988. Also MIT report CFDL-TR-88-3.

(11) Shapiro, R.A. "An Adaptive Finite Element Solution Algorithm for the Euler Equations." MIT PhD Thesis, Dept. of Aeronautics and Astronautics, May 1988. Also MIT report CFDL-TR-88-7. To appear as a Volume in NOTES ON NUMERICAL FLUID MECHANICS, Vieweg.

(12) Kallinderis, J.G. and Baron, J.R. "Unsteady and Turbulent Flow Using Adaptation Methods", pp 326-330, Lecture Notes in Physics, Vol. 323, 11th Int. Conf. on Numerical Methods in Fluid Mechanics, Editors: D.L.Dwoyer, M.Y.Hussaini, and R.G.Voight, Springer-Verlag, 1989.

(13) Shapiro, R.A. and Murman, E.M. "Higher-Order and 3-D Finite Element Methods for the Euler Equations." AIAA paper 89-0655, January 1989.

(14) Baron, J.R. "Embedded Adaptation for Time Dependent Real Fluid/Gas Flows", pp 17-22, Proceedings of 7th Int. Finite Element Conference on Flow Problems, Univ. of Alabama at Huntsville. Editors: T.J.Chung and G.R.Karr, 1989.

(15) Kallinderis, I., "Adaptation Methods for Viscous Flows". MIT PhD thesis, May 1989. Also MIT report CFDL-TR-89-5, May 1989.

(16) Kallinderis, J. and Baron, J.R., "The Finite Volume Approach for the Navier-Stokes Equations". To appear as chapter in COMPUTATIONAL METHODS IN VISCOUS AERODYNAMICS. Editor: C.A.Brebbia, Springer-Verlag 1989.

(17) Kallinderis, J. and Baron, J.R., "Adaptation Methods for Viscous Flows". To appear as chapter in COMPUTATIONAL METHODS IN VISCOUS AERODYNAMICS. Editor: C.A.Brebbia, Springer-Verlag 1989.

(18) Kallinderis, Y. and Baron, J.R., "An Adaptive Algorithm for Turbulent Flows". Submitted to J.AIAA, June 1989.

(19) Kallinderis, Y. and Baron, J.R., ""Application of an Adaptive Algorithm to Single and Two-Element Airfoils in Turbulent Flow". Submitted to AIAA Aerospace Sciences Meeting for Reno in January 1990.

(20) Landsberg, A. and Murman, E., "Implementation of a 2D-Euler Solver Using Id World, a Parallel Programming Environment," AIAA paper 89-1941, 9th Computational Fluid Dynamics Conference, Buffalo, NY, June 1989.

PROFESSIONAL PERSONNEL ASSOCIATED WITH RESEARCH EFFORT

Principal Investigators

Earll M. Murman
Judson R. Baron

Others

Saul S. Abarbanel	Senior Lecturer, MIT; Professor, Tel-Aviv University
John G. Kallinderis	PhD Candidate
Bernard Loyd	PhD Candidate
David Modiano	SM Candidate
Richard A. Shapiro	PhD Candidate
Alexandra Landsberg	SM Candidate
Jorge Perez-Sanchez	SM Candidate
Robert Haimes	Research Associate

INTERACTIONS

Papers presented:

- AIAA 87-1111 presented at AIAA 8th Computational Fluid Dynamics Conference, Honolulu, June 1987.
- AIAA 87-1167 presented at AIAA 8th Computational Fluid Dynamics Conference, Honolulu, June 1987.
- AIAA 88-0034 presented at AIAA 26th Aerospace Sciences Meeting, Reno, NV, Jan. 1988.
- Presentation at 7th GAMM Conference on Numerical Methods in Fluid Mechanics, Sept. 1987.
- Presentation at 2nd International Conference on Hyperbolic Problems, Aachen, Germany, March 1988.
- Presentation at 3rd International Conference on Supercomputing, Boston, MA, May 1988.
- Presentation at 11th International Conference on Numerical Methods in Fluid Dynamics, Williamsburg, VA, June 1988.
- AIAA 89-0655 presented at the 27th Aerospace Sciences Meeting, Reno, NV, Jan. 1989.
- Presentation at 7th Int. Finite Element Conference on Flow Problems, University of Alabama at Huntsville, April 1989.
- AIAA 89-1941 presented at the AIAA 9th Computational Fluid Dynamics Conference, Buffalo, NY, June 1989.

Seminars:

- "Multi-Topics for Multistage Methods", presented at Air Force Wright Aeronautical Laboratories, Nov. 13, 1987.
- "Computational Fluid Dynamic Simulations on a Mini Supercomputer," Sigma Xi meeting, Tullahoma, TN, Nov. 17, 1987.
- "Adaptive Computations in Space and Time", presented at University of Maryland, College Park, November 1988.

Other:

- MITOSIS code software developed for embedded adaptation algorithm under predecessor grant has been licensed to NASA Ames Research Center.

NEW DISCOVERIES

None

APPENDIX /

AIAA 87-1111-CP

A Hybrid Expert System for Complex CFD Problems

John F. Dannenhoffer, III
Judson R. Baron

Massachusetts Institute of Technology
Cambridge, MA

AIAA 8th Computational Fluid Dynamics Conference
June 9–11, 1987 / Honolulu, Hawaii

AIAA 87-1111-CP
Honolulu, June 1987

A Hybrid Expert System for Complex CFD Problems

John F. Dannenhoffer, III [*]

Judson R. Baron [†]

Computational Fluid Dynamics Laboratory
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, MA

Abstract

A new problem solving strategy for complex CFD systems is described, revolving around a new hybrid system which incorporates the advantages of both conventional and expert systems. The paper describes the structure of expert systems and subsequently their operation and illustrates the technique through a local compressible flow analysis example. Comparison of the operations for forward- and backward-chaining expert systems with conventional systems leads to an understanding of the advantages and disadvantages of each. Based upon these observations, a new hybrid system and associated problem solving strategy is described. Application has been made to complex grid adaptation of the two-dimensional Euler equations, highlighting the high efficiency and flexibility of the hybrid system.

Introduction

Recently, a great deal of attention has been focused on expert systems, both because of their power and their mystique. Early systems, most notably MYCIN[1] (for diagnosing bactermia and meningitis infections) and DENDRAL[2] (for predicting the molecular structure of compounds from mass spectrograms), all dealt with expert domains in which conventional systems (for example FORTRAN programs) have not been successful. This led some to make the exaggerated claim that expert systems are sufficiently powerful so as to replace conventional systems in the future. However despite their utility as a very powerful tool for some applications, their direct use for most CFD problems would probably be unwise due to their very low efficiency at repetitive numerical tasks.

*Research Assistant, Member AIAA
†Professor, Associate Fellow AIAA

To circumvent this difficulty, several CFD researchers have combined expert system and conventional techniques, yielding hybrid systems. Three such efforts are described here.

The first of these is the work of Tong[3] in which he uses an expert system to automate the design process for cooling fans. His system basically consists of two components: a conventional system to analyse a proposed geometry and an expert system to alter the geometry based upon the observed performance of the previous geometry and heuristic design rules gleaned from a human design expert. Successive iterations between these two components results in the evolution of an improved cooling fan.

The second example is the PAN AIR knowledge system by Conner and Purdon[4]. Here there are two major components: an expert system with which a user consults in order to set up a proper input file for the second component, the PAN AIR program which is written using conventional techniques.

The third example is a zonal grid generation system discussed briefly by Andrews[5]. Here an expert system is used to divide complex two-dimensional flow fields into four-sided, well-shaped zones; a grid generation program (which uses conventional programming techniques) then generates the final computational grid within each zone.

In each of these systems, the conventional and expert system components are rather loosely coupled, that is there is a limited interaction between the two components. Conner et al and Andrews apply the two components successively with the expert system performing the initial processing and then conventional components acting on the expert system's outputs. Even in Tong's work, where he alternately applies the two types of components, the components act somewhat independently.

In this paper a tightly coupled use of conventional and expert system components is described. This new hybrid system architecture gains its power by combining the advantages of each type of component at a lower level.

This paper begins by describing expert systems, both in terms of their components and their operation. Through a case study, local compressible flow analysis, the operation of an expert system is contrasted with the operation of a conventional system, making the advantages and disadvantages of each apparent. This then leads to the development of the general hybrid system architecture which forms the framework for the MITOSIS adaptive grid program[6]. The paper concludes with a full description of the hybrid system's knowledge base that is used to accomplish adaptation, both as initially set up and through various revisions to demonstrate the ease with which experience can be built into the program.

Expert Systems — Background

Expert systems are computer systems which approach problems that are normally associated with human experts by applying a *reasoning* mechanism to a body of *knowledge* in its domain.

As shown in figure 1, all computer systems, whether expert or conventional, are composed of three basic components:

Facts — descriptions of the state of a physical or abstract object or situation of interest. For example, the radius of a circle or an employee's name.

Relationships — statements concerning the interconnections of facts. For example, the circumference of a circle is related to its radius by the algebraic relation $c = 2\pi r$.

Control — statements concerning the order in which the various relationships should be applied to specific facts in order to accomplish a task, typically the latter being the creation of new facts.

In conventional systems, programs consist of two major components: the *variables* which contain the facts and the *program* which contains both the relationships and control. Typically relationships are coded as assignment statements while control is coded implicitly through statement ordering and explicitly as GOTOs, DO-loops, and subroutine calls. The essential point is that the relationships and control are intertwined.

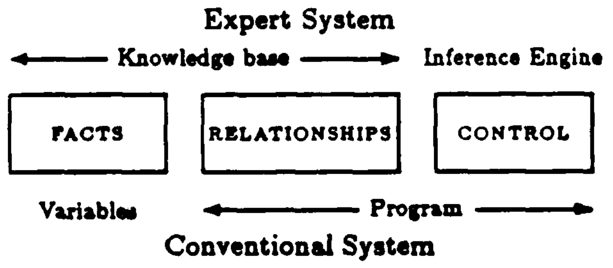


Figure 1: System organisation

In expert systems on the other hand, the rules are combined with the facts in the *knowledge base* and control is in a separate component known as the *inference engine*. In this way, the domain dependent information (facts and rules) is segregated from the domain independent information (control).

The facts in an expert system, not unlike variables in conventional programs, have names and values such as `airfoil-type:NACA0012` and `Mach-number:0.80`. As shown here, the values can be symbolic or numeric or can have the value `UNKNOWN`. The operation of many expert systems relies heavily on allowing values to be unknown, and being sought as part of the solutions. In addition to this simple representation of facts, other expert systems include more complicated representations such as object-attribute-value triplets, frames, and semantic nets[7].

The relationships in an expert system describe cause-and-effect or other relationships between facts in the domain, known as *rules*. Expressed as if-then statements, rules typically take the form:

```
Rule:
        if:         premise clause 1
        andif:      premise clause 2
                        :
        then:       action clause 1
        andthen:    action clause 2
                        :
```

In general, *premise clauses* are simple relational operators which evaluate to either TRUE, FALSE, or UNKNOWN. In order for a rule to be available for execution, it cannot contain any false premise clauses.

The *action* part of a rule consists of one or more action clauses, each of which either modifies facts in the knowledge base, performs a user-defined function, or performs input and/or output. In some expert systems, it is even possible to create new rules or disable old ones. The standard functions which are supported by most expert systems include the arithmetic operations as well as other common mathematical functions such as root extraction. User-defined functions are generally just implementation-language functions which are executed based upon facts supplied through the rules and which generate new facts that are stored in locations specified by the rule.

The control in an expert system is done in the *inference engine*, whose responsibility it is to decide which rules in the knowledge base are to be *fired* (executed) and in what order; in other words to reason with the knowledge (the facts and relationships). There are two basic strategies used in inference engines: forward- and backward-chaining.

In forward-chaining, which is also known as data-driven inferencing, the rules are applied to the currently known facts such that all subsequently determinable facts are found. This is accomplished through the following steps:

search — search through all rules, listing only those which have true premises. If none qualify, execution of the inference engine halts.

select — if the search list contains more than one entry, select the most dominant entry from the list. This step is also known as *conflict resolution*.

act — perform the actions in the action clauses of the most dominant rule on the list, resulting in a modified set of facts in the knowledge base. The process of executing action clauses in a rule is also known as *firing* the rule.

repeat — return to the search step

The Digital Equipment Company uses an expert system called XCON[8] for custom configuring VAX computer systems. This synthesis task, in which component compatibility is checked, power supplies and cabinets are assigned, and cables are designed, is ideally suited for a forward-chaining inference engine.

In backward-chaining, which is also known as goal-driven inferencing, the procedure is inverse and begins with a goal. The rules are applied backwards and those which assert a value for the goal are selected. If the rules themselves reference facts with currently unknown values, these facts become new goals and the processing continues recursively. This type of inferencing strategy is particularly appropriate for classification and diagnostic problems, One of the first true expert systems, MYCIN[1], is a backward-chaining system whose domain of expertise is diagnosing bactermia and meningitis infections.

In both chaining systems, the inferencing mechanism can be summarised by *match-select-act-repeat*. The match step searches through all of the rules, seeking those which match either current data or goals, the select step uses a conflict resolution strategy to pick a dominant rule from those identified in the match step, the act step performs the actions in the then-clause of the selected rule, and the entire cycle is then repeated.

The examination of all the rules (or a subset of them) in the match step is what gives expert systems their power and at the same time makes them inefficient for repetitive tasks which can be defined procedurally. Acceleration strategies to circumvent this problem (for example, context limiting[9] and the Rete match algorithm[10]) have been developed but unfortunately even their use results in a system which is less efficient for some repetitive tasks than is the corresponding conventional system.

attribute	description
a	speed of sound
A	cross-sectional area of streamtube
$\dot{m}$	mass flow rate
M	Mach number
p	static pressure
R	gas constant
T	static temperature
u	velocity
γ	ratio of specific heats
ρ	static density

Table 1: Initial set of attributes for the local compressible flow analysis model problem

Model Expert Systems

Consider two expert systems, one forward-chaining and one backward-chaining, which use a common knowledge base to attack a model fluid mechanics problem. Though the expertise here consists mainly of simple algebraic substitutions, the examples do highlight the real strengths and weaknesses of expert systems through the comparison of their operation with that of conventional procedural systems.

The domain of expertise of these model expert systems is local compressible flow analysis, that is those aerodynamic and thermodynamic variables describing the relationships governing the flow at a point in a streamtube. Initially the conditions will be characterised by the attributes given in table 1 and the relationships in table 2. It is assumed that the values and relationships are all written in a consistent set of units (or are non-dimensional).

Since there is no provision for algebraic manipulations in the inference engine as used here, each of the relationships given in table 2 must be written as a series of rules, leading to the rule set given in table 3. The attributes γ and R were considered parameters whose values would be assigned *a priori* by the user and hence were not solved for in this set of rules, although one could certainly do that if desired.

The appropriate form of the rule to implement $p = \rho RT$ is given by:

```
Rule 1:  if:
         then:   $p \Leftarrow \rho RT$
```

relationship	description
$p = \rho RT$	state equation
$a^2 = \gamma RT$	speed of sound relation
$\dot{m} = \rho u A$	mass flow rate definition
$M = \frac{u}{a}$	Mach number definition

Table 2: Initial set of relationships for the local compressible flow analysis model problem

The lack of premises in rule 1 is tied to the lack of restrictions in using the relationship $p = \rho RT$; physically the only restriction for using this rule is that the fluid is thermally perfect. If one wanted to extend the present knowledge base to include thermally imperfect gases, then physical restrictions would show up as premise clauses in the rules. As a result, rule 1 would be written as the two rules:

```
Rule 1a:  if:     gas is perfect
          then:   $p \Leftarrow \rho RT$

Rule 1b:  if:     gas is imperfect
          then:   $p \Leftarrow \cdots$
```

This illustrates one of the real powers of expert systems, that is, that the rules in the knowledge base should be concerned with representing the physics and not the control flow. Concerns about whether or not a rule applies to the current situation (except for physical restrictions embodied in premise clauses) are addressed by the controlling mechanism in the inference engine.

It is instructive now to examine the operation of the *forward-chaining inference engine* when applied to the knowledge base given above in tables 1 and 3. Since the order of rule application is heavily dependent on the data, two cases will be examined, their difference being the initial set of known facts.

rule number	rule statement
1	$p = \rho RT$
2	$\rho = \frac{p}{RT}$
3	$T = \frac{p}{\rho R}$
4	$a = \sqrt{\gamma RT}$
5	$T = \frac{a^2}{\gamma R}$
6	$\dot{m} = \rho u A$
7	$\rho = \frac{\dot{m}}{uA}$
8	$u = \frac{\dot{m}}{\rho A}$
9	$A = \frac{\dot{m}}{\rho u}$
10	$M = \frac{u}{a}$
11	$u = Ma$
12	$a = \frac{u}{M}$

Table 3: Initial set of rules for the local compressible flow analysis model problem

In the first case, the attributes T, R, γ, M, $\dot{m}$, and A are assumed to initially have known values, yielding the locus of knowledge illustrated in figure 2. Facts are represented by circles containing the attribute name and rules (which are numbered) are represented by groups of vertical and diagonal links between the circles. For example the three links emanating from the top of a in the figure indicate that a can be asserted using rule 4 given known values of T, R, and γ.

The initial set of facts are shown across the top of the figure in a somewhat arbitrary order. As a result of the search step in the first cycle, only rule 4 was triggered (met the search criteria) to yield the value of a, the speed of sound. Thus after the first cycle, the parameters with known values are those shown on the top two rows of the figure. In like manner, the second cycle resulted in a value for u through the triggering and firing of rule 11. This figure clearly shows that in order to determine the value of ρ from the initially known attributes with the original set of rules, one must first compute values for a and then u.

In the second example, the initial set of facts consists of M, γ, R, T, p, and A; the resulting the locus of knowledge

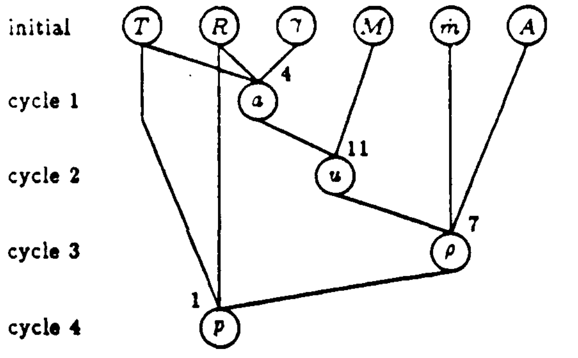


Figure 2: Locus of knowledge for first forward-chaining example

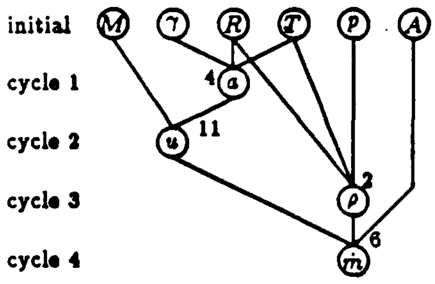


Figure 3: Locus of knowledge for second forward-chaining example

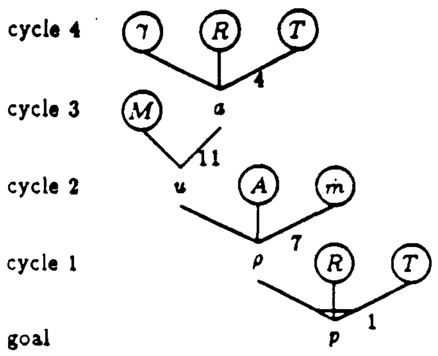


Figure 4: Locus of knowledge for backward-chaining example

(shown in figure 3) differs significantly from the previous case.

In summary, both forward-chaining expert systems start with an initial set of facts and then use the rules to assert all possible consequent facts. The exact locus of knowledge is heavily dependent on the initial facts and to a somewhat lesser degree on the conflict resolution strategy chosen.

Consider now a backward-chaining inference engine applied to the local compressible flow knowledge base with the same initial set of known facts as in the first example above (T, R, γ, M, $\dot{m}$, and A). For an inference engine invocation with p as the goal, the locus of knowledge is given in figure 4. Here facts with known values are denoted by an attribute name in a circle, while those whose values are UNKNOWN are not circled. Rules are denoted in the same way as in previous knowledge loci.

For this case, the original goal (p) is shown on the bottom row of the figure. The search step in the first backward-chaining cycle finds rule 1 which asserts a value for p. However in order to fire, a value must be known for ρ. Therefore ρ is made the new goal and the process repeats. At this point there are two rules, 2 and 7, but rule 2 must be excluded since it requires a value for p which is already a goal. The cycling continues until rule 4 is found as a means of calculating a value for a. Since all its inputs are known, it can be fired, making the value of a available so that rule 11 fires, ... Eventually the value for the original goal p is determined.

One of the particularly nice features of expert systems is that the separation of knowledge from the control makes it relatively easy to add new knowledge and hence new capability. For example in the local compressible flow example, stagnation pressure (p_o) and stagnation temperature (T_o) are easily added simply by defining two new attributes and by adding new rules. These new rules, which are listed in table 4, are derived from the energy equation and isentropic relations.

rule number	rule statement
13	$T_o = T\left[1 + \frac{\gamma-1}{2}M^2\right]$
14	$T = T_o\left[1 + \frac{\gamma-1}{2}M^2\right]^{-1}$
15	$M = \left[\frac{2}{\gamma-1}\left(\frac{T_o}{T} - 1\right)\right]^{\frac{1}{2}}$
16	$p_o = p\left(\frac{T_o}{T}\right)^{\frac{\gamma}{\gamma-1}}$
17	$p = p_o\left(\frac{T}{T_o}\right)^{\frac{\gamma}{\gamma-1}}$
18	$T_o = T\left(\frac{p_o}{p}\right)^{\frac{\gamma-1}{\gamma}}$
19	$T = T_o\left(\frac{p}{p_o}\right)^{\frac{\gamma-1}{\gamma}}$

Table 4: Rules added to the local compressible flow analysis model problem

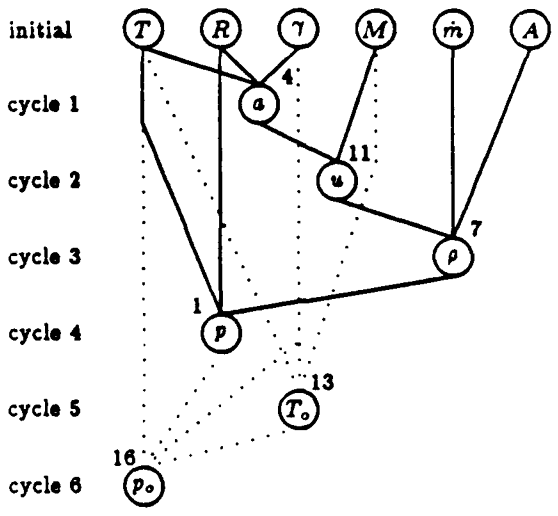


Figure 5: Locus of knowledge for forward-chaining example with extended knowledge base

With this expanded set of rules, the initial set of facts from figure 2 now produces the locus of knowledge shown in figure 5, where the dotted lines simply denote the newly added rules. It should be noted that the searches in cycles 1 through 4 all contained two rules in their conflict set, the rule which fires and rule 13; the firing of rule 13 was arbitrarily delayed until cycle 5 by the conflict resolution strategy.

Comparison of Expert and Traditional Systems

The previous sections described expert systems both in terms of their components and through the examination of their operation when applied in various ways to a model problem. Now consider the characteristics of expert systems in contrast to traditional procedural systems in an attempt to elucidate the strengths and weaknesses of the two approaches.

Virtually all of the differences between expert and procedural systems grow out of their structural differences. Recall that expert systems consist of two parts: a knowledge base containing both facts (attributes) and relationships amongst the facts (rules), and a control strategy (inference engine) which models one of a few general problem solving strategies. On the other hand, procedural systems are segmented differently: the variables (data) and the program which includes both relational and control information.

One implication of this structural difference is that the separation of rules from control in expert systems allows relationships to be written in the form of cause-and-effect statements which directly represent physical laws and/or heuristics. As a result, the domain knowledge is explicitly stated in the rules, and modifications to the underlying physical model or extensions to the system's domain of expertise is more easily accomplished with an expert system than is possible with a traditional procedural system in which the relationships are intermixed with control.

A side benefit is that since the knowledge is explicitly represented in the rules, it is relatively straightforward to trace the locus of knowledge through the expert system. Sometimes this is even done automatically by including an explanation facility in the inference engine where the user can ask questions such as "why is a value required for this attribute?" or "how was this conclusion reached?". The explanation facility answers such questions by tracing through the rule and/or goal stacks created internally by the inference engines or by tracing through a list of rules which were fired.

Another important implication of the separation of relationships and control in an expert system is that the controller (inference engine) can directly mimic one of the problem solving strategies used most frequently by experts. Just as human experts sometimes attack a problem forward from the initial set of facts, using all appropriate physical laws and heuristics, so too will a forward-chaining inference engine work forward, choosing the appropriate rules to develop all consequences of the initial data. On the other hand, problems in which human experts work backward through physical laws and heuristics in search of values for a particular goal are well suited to backward-chaining inference engines. Of course, for efficiency some domains require a combination of the two basic strategies (perhaps applied iteratively), resulting in inference engines which combine elements of both forward- and backward-chaining.

The essential point is that either type of inference engine applies only those rules which are appropriate to the current situation, ignoring those in the knowledge base which are currently superfluous. Thus one can start with virtually any set of facts and a forward-chainer will adapt its application of the rules to match the given initial set. Similarly, a backward-chainer will adapt its search through the rules based upon the desired goal as well as the initially prescribed data. In summary, expert systems do not require *a priori* knowledge of which parameters are inputs and which are outputs.

An unfortunate consequence of this adaptiveness is that for tasks with known sets of input and outputs, expert systems are slow relative to procedural systems. This is due primarily to the search which has to be undertaken in each cycle on the inference engine. Additionally, tasks which can be described procedurally, that is tasks which follow a predefined recipe of operations and flow control based upon simple comparisons, cannot be easily prescribed in an expert system since by design the rules can reference facts but not other rules. Hence for applications which require intensive amounts of repetitive calculations which *can* be defined procedurally, the relative inefficiency of expert systems makes them unsuitable.

It should be noted again that expert and procedural systems are simply alternative strategies for problem solving. Either strategy can be used to solve any problem even though the other may be easier to implement or may be more efficient. In general, problems which require a large amount of repetitive calculations (which can be prescribed algorithmically) are better suited for procedural systems; those for which the domain knowledge grows and/or those in which the types of inputs are not known *a priori* are better suited for expert systems.

A Hybrid Expert/Procedural System

The basic requirements of complex numerical processes, particularly those in Computational Fluid Dynamics (CFD), may be examined in order to determine the relative suitability of expert and procedural systems. It turns out that neither is completely satisfactory by itself, leading to the development of a hybrid system.

As used here, the term *complex numerical process* refers to a procedure which meets the following criteria:

- it is composed of a order ten sub-processes;
- its sub-processes are generally rather complicated, each requiring thousands, millions, or even more operations;

1	preliminary airfoil design	7	2-D design
2	planform design	8	3-D inviscid analysis
3	section extraction	9	3-D viscous analysis
4	section fairing	10	mechanical properties
5	2-D inviscid analysis	11	graphics
6	2-D viscous analysis		

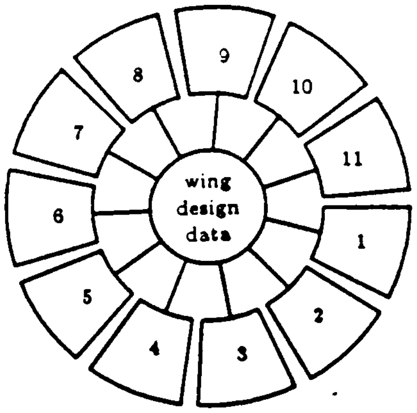


Figure 6: Components of a typical wing design program

- its sub-processes are generally numeric in nature, that is mostly composed of simple arithmetic operations and simple logical tests;
- its sub-processes are relatively independent of each other, that is they may viewed as black boxes with the only communication between them being through a prescribed set of inputs and outputs; and
- the order in which the sub-processes are executed is not fixed but instead changes depending on results from the various sub-processes.

If more than order ten sub-processes are present, one could abstract the sub-processes at a higher level, essentially creating a hierarchy of complex numerical processes.

As an example of a complex numerical process, consider the design of an aircraft wing as depicted in figure 6. At the center of the figure is a large pool of data, containing all geometric and aerodynamic properties which describe the wing and its operation. Data in the central pool might contain the shape of the wing (both planform and sections), freestream flight conditions, spanwise loading distribution, and pressure distributions, etc. Values for many of the properties in the data pool may be unknown at early stages in the design process, awaiting application of the various sub-processes.

Eleven such sub-processes are shown surrounding the central data pool in figure 6. Included in this list are two- and three-dimensional analysis processes, processes to convert two-dimensional to three-dimensional data and vice versa, design modules, and graphics processors. Those included are not meant to form an exhaustive set but are typical sub-processes.

All sub-processes communicate with the central data pool via a two-way link. Additionally, a sub-process cannot communicate directly with any other but instead must deposit information into the central data pool for another sub-process to extract. Though this communication stricture may seem overly severe, it leads to substantial benefits in the design and development of the overall process. First, the function of each sub-process is completely independent of other sub-processes, and can be developed and tested individually. Second, because the inner workings of each sub-process is unknown to the other sub-processes, the algorithm followed can be changed as long as the output remain identical.

The exact sequence of sub-processes used by a wing designer cannot be prescribed *a priori* but instead is highly dependent on the data which is output from prior sub-processes. Very often the choice of which sub-process to employ is heuristic in nature. Economical, political, and technological concerns all impact the choice of sub-processes which the designer chooses. For example, three-dimensional viscous analyses are currently prohibitively expensive and may be used only in the most necessary cases. Similarly, though two-dimensional analyses are relatively inexpensive, their use is somewhat limited by the assumptions made in their development.

The development of a complex numerical process such as for the above wing design example can be accomplished in a variety of ways; here two competing techniques will be explored: procedural and expert systems.

First consider the traditional, procedural system. For the execution of the algorithmically-based sub-processes, the procedural system will be highly efficient, primarily due to the fact that procedural systems compile directly into machine instructions which more or less directly mimics the algorithm.

At the same time, the data-sensitivity necessary to properly determine the order of execution of the sub-processes is difficult to implement very well in a procedural system. For a few sub-processes and a few possible conditions, this can be handled with some IF-tests and GOTOs. However as the number of sub-processes and possibilities increases, the logic quickly become unmanageable, often making extensions nearly impossible to implement. Many large systems have been rendered inoperative (at least temporarily) by adding just one small new feature somewhere else in the system because the reasons why and when certain sub-processes were executed were obscured by the logical structure of the controller.

Second, consider using an expert system. Here the controlling logic concerning the execution order of the various sub-processes is very easy to implement and extend through the use of expert system rules. The logic is easy to decipher because it is represented explicitly in the rules.

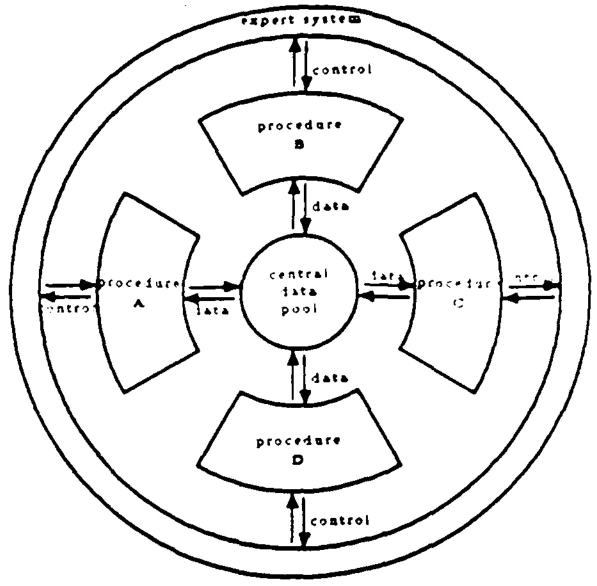


Figure 7: Organisation of hybrid system

Unfortunately the execution of the algorithms in the sub-processes is considerably less efficient in an expert system than in a procedural system, mostly due to the search step in expert systems.

Clearly neither procedural nor expert systems alone is completely suitable for complex numerical processes and hence a new hybrid system is proposed here, as shown in figure 7.

At the center is the data pool which contains all information to be shared amongst the procedures. Typically the data consists of some global parameters which describe the current problem as well as arrays which describe distributions.

Surrounding the data pool are a small number of independent procedures which generally are both numerical and algorithmic in nature, making them ideal candidates for procedural system techniques.

The final component is the expert system (shown in figure 7 as a ring encircling the procedures), the purpose of which is to control the flow of execution through the system. As such it passes control information into and receives status information out of the various procedures.

Both the flexibility and efficiency of the hybrid system compare favorably with that of expert and procedural systems when applied to complex numerical processes. On the one hand, since the majority of computer resources (CPU time) are consumed by the execution of the sub-processes, the efficiency of the hybrid system is essentially the same as the efficiency of procedural systems and much greater than that of an expert system. On the other hand, since the control of the execution order of the sub-processes is governed by rules in the expert system, the flexibility and data sensitivity of the hybrid system is much the same as an expert system and much greater than is possible with a procedural system.

The structure of the hybrid system developed here is different from the hybrid systems of Tong, Conner and Purdon, and Andrews in one important respect — the level at which the expert and conventional components are coupled. In the other three systems, the components are coupled at a very high level, that is a brief conceptual block diagram of the data and processing flows through these hybrid systems would include a very small number of blocks which were conventionally programmed and others which included expert systems. The two types of components are of roughly equal stature in that each contains processing and control functions.

On the other hand, the hybrid system developed here is coupled at a much lower level; the conventional procedures are responsible for all CPU-intensive functions and the expert system serves a purely supervisory function, that is it controls the order of execution of the various processes based upon status information passed back from the processes to the expert system. This separation of processing and control is a major benefit of the new hybrid system approach.

Application of Hybrid System to Grid Adaptation

The effectiveness of the hybrid system approach has been demonstrated by applying it to a two-dimensional grid adaptation program, the MITOSIS system[6]. This section begins with a brief overview of the grid adaptation problem along with typical computed results from the MITOSIS program. A discussion of how MITOSIS fits into the hybrid system strategy follows. The section concludes with a description of the initial MITOSIS knowledge base as well as additions which were included as experience accumualted through the use of the MITOSIS system.

The Grid Adaptation Problem

The conflicting requirements of accuracy and efficiency dictate that advanced calculation methodologies be employed for complex flow-fields. One method of coping with the accuracy/efficiency dilemma is to use an adaptive grid technique such as MITOSIS in which an Euler flow is solved using a Lax-Wendroff-type integration scheme on a succession of grids. For simple cases, the scheme consists of the following steps:

1. initialize the fixed, coarse global grid using any standard grid generation method
2. integrate the Euler equations forward in time on the current grid until the residual has fallen to some specified level and until the global convergence parameters (such as lift and drag coefficients) have stabilized
3. stop if the converged solution is sufficiently close to the converged solution on the previous grid, otherwise
4. search the flow field for local flow features, that is regions where the changes in flow-field quantities are large (for example shocks and stagnation regions)
5. divide cells near flow features and fuse (un-adapt) cells away from flow features, so as to concentrate the computational grid where required
6. go back to step 2

More complicated cases require a slightly different set of steps as descibed below.

Figures 8 and 9 show the original and final (after five adaptation cycles) computational grids for an RAE-2822 airfoil at a free-stream Mach number of 0.75 and three degrees angle of attack. The differences in the Mach number distributions for these two cases can be seen clearly in figure 10, indicating that the increased resolution, at least in the vicinity of the shock, is required for this case. The Mach number distribution obtained with the adapted grid is the same as that obtained using a globally refined grid, but required about 100 times less computer time.

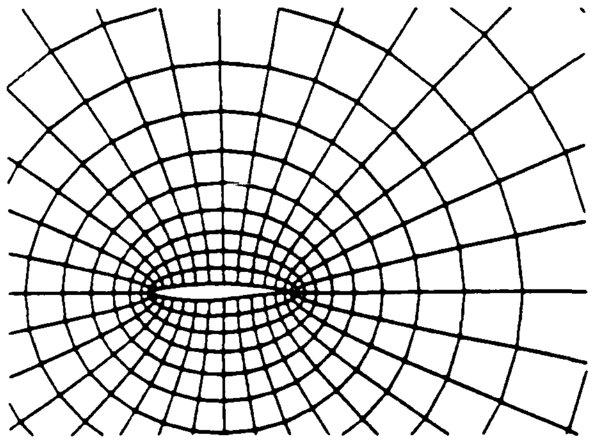


Figure 8: Initial computational grid, RAE-2822, $M_\infty = 0.75$, $\alpha = 3°$.

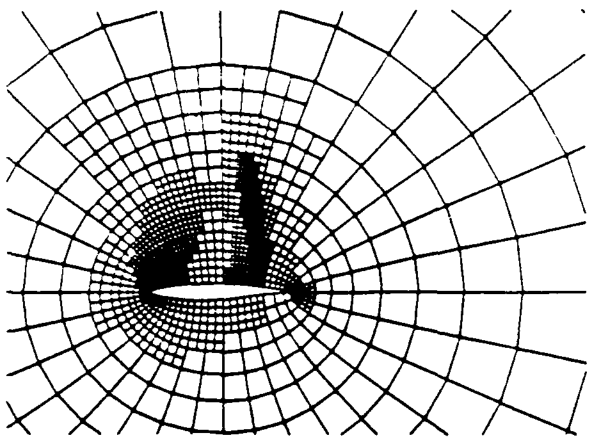


Figure 9: Final adapted computational grid, RAE-2822, $M_\infty = 0.75, \alpha = 3°$.

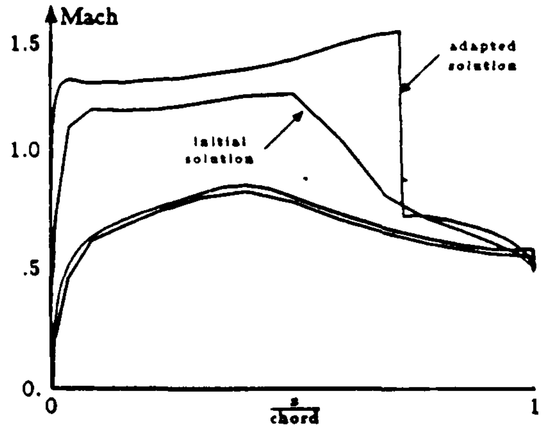


Figure 10: Mach number distributions (before and after adaptation), RAE-2822, $M_\infty = 0.75, \alpha = 3°$.

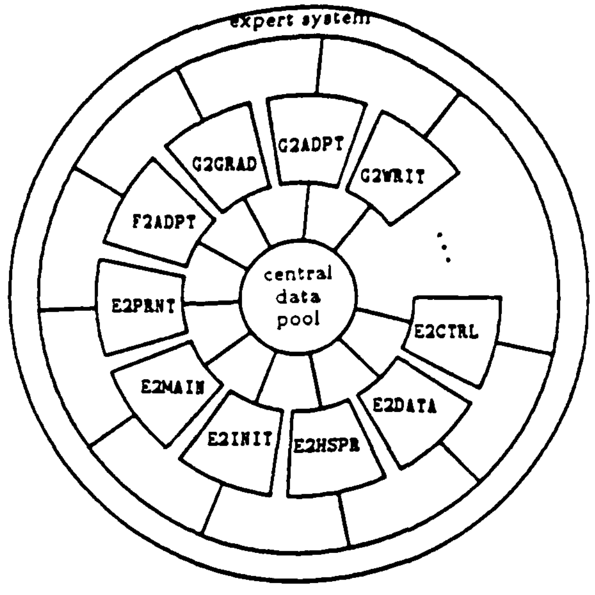


Figure 11: Organisation of MITOSIS

MITOSIS as a Hybrid System

The application of the hybrid system strategy to MITOSIS is shown schematically in figure 11. As with all hybrid systems, the central data pool is surrounded by *application procedures*, which in turn are surrounded by the expert system whose responsibility it is to schedule (control) the execution of the procedures. A brief description of each of these components follows; a complete discussion is contained in [11].

The center of the figure contains the central data pool in which data which is to be communicated amongst the procedures is stored. For MITOSIS this includes nodal arrays containing the dependent and independent variables throughout the flow field, data structure arrays which specify the interconnections of the nodes as well as information used within the Euler integrator, problem parameters (such as free-stream reference values, smoothing coefficients), and convergence information (for example, iteration number, normalised changes in the dependent variables from the previous iteration, etc.)

The procedures wh'ch surround the central data pool are those which consume the vast majority of the computational resources. The procedures in the figure, which are written using conventional programming techniques, include E2CTRL (the Euler integrator initialisation routine), E2MAIN (the procedure which actually integrates the Euler equations one iteration), F2GRAD (the procedure which computes the first difference of the refinement parameter), and G2WRIT (the procedure which writes the current data structure to a disk file).

The expert system is responsible for scheduling the execution of the various application procedures. The inferencing mechanism used here is forward-chaining with the conflict resolution strategy being such that the rule with the greatest number of premise clauses dominates.

The Initial MITOSIS Knowledge Base

MITOSIS' knowledge base consists of two parts: the facts and the rules. The facts contain constants (the input/output units of the printer, etc.) as well as those which vary through the execution of the inference engine. Amongst these are the current convergence status (conv-check), the lift coefficient (clift), the division threshold (thr-divd), and the number of adapted cells (ncells).

The initial MITOSIS knowledge base consisted of 10 rules which are described in [11]; following is an abridged list which serves to illustrate the controlling role of the expert system. The rules are formatted such that lower-case words represent variables and upper-case represents fixed control words. As with all computer processing, it is advantageous to break the control problem into a series of smaller tasks, herein called *contexts*. One could consider these contexts to serve somewhat the same purpose as subroutines and procedures in conventional computer systems. The MITOSIS knowledge base initialises the context to initial.

Each time that a new grid is established (either initially or after an adaptation), various initialisations are required. These are the subject of the first rule:

```
RULE     SET CONTROL AND INDEX VECTORS
IF       context .EQ. "initial"
THEN     E2CTRL
ANDTHEN  G2SUMY terminal
ANDTHEN  E2STAT printer
ANDTHEN  SET iters +0.0
ANDTHEN  SET conv-stage +0.0
ANDTHEN  SET conv-check +0.0
ANDTHEN  SET context "integrate"
```

The actions performed by this rule are: set up the control and index vectors for the Euler integrator (E2CTRL); output the grid and solver status information (G2SUMY and E2STAT); and initialise the attributes iters, conv-stage, and conv-check. The rule concludes by switching the context to integrate. Since this is the only rule associated with context initial, it is always the first rule to fire.

Rules with the context integrate are associated with integrating the Euler equations to steady state on a given grid. There are four rules in this context, one each for the four major types of processing which is to be done in this context. The first rule in this context:

```
RULE     TAKE ONE ITERATION
IF       context .EQ. "integrate"
THEN     E2MAIN delta-du clift cdrag
ANDTHEN  E2WIND conv-stage conv-check
ANDTHEN  E2HSPR
ANDTHEN  ADD iters iters +1.0
```

is the one which calls E2MAIN to integrate the Euler equations one iteration. It then calls the convergence checking algorithm (E2WIND), outputs convergence information for the iteration just taken (E2HSPR), and increments the number of iterations taken on this grid.

The second rule in context integrate is:

```
RULE     CONVERGENCE DETECTED
IF       context .EQ. "integrate"
ANDIF    conv-check .EQ. +1.0
THEN     SET context "converged"
```

and is responsible for noting that the Euler integrator has converged on the current grid by monitoring the value of conv-check which is updated each time the rule discussed previously is fired. Upon noting convergence, this rule switches the current context to converged. This will always dominate the first rule in this context by virtue of the fact that it has two premises whereas the other has only one.

There are rules similar to the latter which are concerned with detecting that the maximum number of allowable iterations on any one grid has been exceeded, as well as that the iteration process on the grid has diverged. In each such case the solution file is dumped to a disk file and the processing halts.

Another context, adapt, consists of three rules which are associated with the grid adaptation process. In like manner to the rules in context integrate, the proper operation of the rules in this context requires that the rule with the most premises dominates in the conflict resolver.

The first rule in this context is re[illegible] for adapting the grid, and is given by:

```
RULE     ADAPTATION WITH AUTOMATIC THRESHOLDS
IF       context .EQ. "adapt"
THEN     ADD levels levels +1.0
ANDTHEN  E2DATA density
ANDTHEN  F2GRAD
ANDTHEN  F2THRS thr-divd thr-fuse
ANDTHEN  F2ADPT thr-divd thr-fuse ncells
ANDTHEN  SHOW ncells
ANDTHEN  SET context "initial"
```

This rule begins by incrementing levels, the number of grid adaptation levels. It then calls E2DATA to write a copy of the density into the work array which F2GRAD subsequently uses to set up the refinement parameter (the first-difference of density). The division and fusion thresholds which are computed next (by F2THRS) are then passed into F2ADPT which actually adapts the grid. The execution of this rule completes by outputting ncells and switching the context to initial to start another cycle.

The determination that the solutions on successive grids are essentially the same and thus further adaptation is not required is the purpose of the next rule, given by:

```
RULE     FINAL CONVERGENCE REACHED
IF       context .EQ. "adapt"
ANDIF    delta-cl .LE. clift-tol
ANDIF    delta-cd .LE. cdrag-tol
THEN     G2WRIT final-soln
ANDTHEN E2PRNT
ANDTHEN SET context "return"
```

When this rule fires, it writes the final solution to unit final-soln and switches the context to return for which there are no rules and thus the inference engine stops.

The final rule in this context ensures that the number of levels of adaptation does not exceed some specified maximum.

These rules simply describe the adaptation process enumerated above. As can be seen from these rules, the logic associated with the grid adaptation strategy is represented in a straightforward way. Tracing the order of execution of the various processes is simplified by the logic being clearly described by the rule text.

Building in Experience

In MITOSIS the philosophy was to develop a simple adaptation strategy which worked well in the majority of cases, and then to add new rules to handle those situations which failed with access to only the initial knowledge base. The ease with which expert systems are easily expanded as knowledge about the domain is identified is a major reason for using expert systems for control and hence the hybrid system approach.

The first situation which was discovered for which the initial knowledge base was insufficient involves the solution for cases where global parameters such as lift and drag coefficients are not available. Here the rule to determine if global convergence is achieved is clearly not applicable. To solve this problem, two new rules were required:

```
RULE     ADAPTATION WITH AUTOMATIC THRESHOLDS(2)
IF       context .EQ. "adapt"
ANDIF    delta_cl .LE. clift_tol
ANDIF    delta_cd .LE. cdrag_tol
ANDIF    clift .EQ. +0.0
ANDIF    cdrag .EQ. +0.0
ANDIF    levls .LT. max_levls
THEN     ADD levls levls +1.0
ANDTHEN E2DATA density
ANDTHEN F2GRAD
ANDTHEN F2THRS thr_divd thr_fuse
ANDTHEN F2ADPT thr_divd thr_fuse ncells
ANDTHEN SHOW ncells
ANDTHEN SET context "initial"
```

and:

```
RULE     FINAL CONVERGENCE REACHED(2)
IF       context .EQ. "adapt"
ANDIF    delta_cl .LE. clift_tol
ANDIF    delta_cd .LE. cdrag_tol
ANDIF    clift .EQ. +0.0
ANDIF    cdrag .EQ. +0.0
THEN     G2WRIT final_soln
ANDTHEN E2PRNT
ANDTHEN SET context "return"
```

These rules are very similar to two rules in the initial knowledge base, except that they contain additional premise clauses. Therefore, if clift and cdrag are both zero, these rules will be found during the forward-chainer's search step, and will dominate since they contain more premises than the corresponding rules in the initial knowledge base.

The second case that required a new rule concerned problems with bow shocks. The difficulty was linked to the fact that the shock's upstream motion that results after the leading edge grid is refined by adaptation passes through the edge of the embedded region. As a result, conservation errors are introduced by the approximations in the embedded mesh formulation. To remedy this problem, a new rule was added:

```
RULE     GROW EMBEDDED REGIONS
IF       context .EQ. "diverged"
THEN     G2READ out-file
ANDTHEN E2DATA density
ANDTHEN F2GRAD
ANDTHEN F2THRS thr-divd thr-fuse
ANDTHEN F2ADPT thr-divd thr-fuse ncells
ANDTHEN G2GROW
ANDTHEN SHOW ncells
ANDTHEN SET context "initial"
```

This rule's action clauses specify that after reading the previous good solution from disk (G2READ), perform simple adaptation, and then grow the embedded regions (G2GROW) so as to avoid (or lessen the chance of) the bow shock penetrating the edge of the embedded region.

The creation of the rules resulted from the following scenario. For each new (different) case that was executed using the then-current version of the MITOSIS knowledge base and as a result of problems that were encountered, an engineer/CFD researcher determined the cause of the problem and fashioned a remedy. The appropriate new rule(s) were then added to the knowledge base (without any modification of the MITOSIS program itself) and the case was re-run. By following this process, the initial set of 10 rules has been expanded to over 25, greatly expanding the robustness of the adaptation process for a wide variety of flow field topologies.

It is largely because of the way in which an expert system functions that adding such new rules proves to be a strightforward process. In principle, this could be implemented with logic in a conventional porogram (as the first version of MITOSIS in fact did). However, the addition of knowledge to conventional structures results in extremely cumbersome codes which are unwieldly to maintain and difficult to understand.

Summary

- A new hybrid system is proposed to treat complex numerical procedures. It consists of the three essential components:
  - a central data pool containing all problem data;
  - any number of procedural elements (written in conventional programming languages) in which the vast majority of computational resources are consumed; and
  - an expert system which schedules and supervises the execution of the various procedural elements.
- The hybrid system is a powerful programming strategy because it is:
  - expandable – new rules concerning the scheduling of the procedural elements for new problems can be easily added;
  - efficient – the vast majority of computer resources are consumed in the procedural elements and therefore the efficiency corresponds to that of procedural systems.
  - understandable – knowledge concerning the scheduling of the procedures is explicitly stated in the rules, making traces easier to understand.
  - maintainable – since procedures are only connected through the central data pool, extensions and maintenance are simplified.
  - dynamic – the system adapts to new input sets and output sets, scheduling only those procedures which are essential.
  - comprehensive – the expertise of more than one expert can be combined, making each user effectively more capable.
- The hybrid system approach is quite effective, as demonstrated by its application to the MITOSIS adaptive grid program.

Acknowledgements

This work was supported by AFOSR Grant 82-0136, Dr. James D. Wilson technical monitor.

References

[1] Buchanan, B.G., and Shortliffe, E.H. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, MA: Addison-Wesley Publishing Company, 1984.

[2] Lindsay, R.K., Buchanan, B.G., Feigenbaum, E.A., and Lederberg, J. *Applications of Artificial Intelligence for Chemical Inference: The DENDRAL Project*. New York:McGraw-Hill, 1980.

[3] Tong, S.S., "Design of Aerodynamic Bodies Using Artificial Intelligence/Expert System Technique", AIAA-85-0112, January 1985.

[4] Conner, R.S. and Purdon, D.J., "PAN AIR Knowledge System", AIAA-86-0239, January 1986.

[5] Andrews, A.E., "Progress and Challenges in the Application of Artificial Intelligence to Computational Fluid Dynamics", AIAA-87-0593, January 1987.

[6] Dannenhoffer, J.F., and Baron, J.R., "Robust Grid Adaptation for Complex Transonic Flows", AIAA-86-0495, January 1986.

[7] Harmon, P., and King, D., *Expert Systems — Artificial Intelligence in Business*, New York:John Wiley and Sons, Inc., 1985

[8] Kraft, A. "XCON: An Expert Configuration System at Digital Equipment Corporation" In *The AI Business: The Commercial Uses of Artificial Intelligence*, edited by P. H. Winston and K. A. Prendergast, Cambridge, MA:The MIT Press, 1984.

[9] Winston, P.H., *Artificial Intelligence*, Reading, MA: Addison-Wesley Publishing Company, 1984.

[10] Brownston, L., Farrell, R., Kant, E., and Martin, N., *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*, Reading, MA: Addison-Wesley Publishing Company, 1986.

[11] Dannenhoffer, J.F., "Grid Adaptation for Complex Two-Dimensional Transonic Flows", Sc.D. thesis, Massachusetts Institute of Technology, Cambridge, MA. (in preparation)

APPENDIX 2

AIAA 87-1167-CP

Adaptation Methods for a New Navier-Stokes Algorithm

John G. Kallinderis
Judson R. Baron

Massachusetts Institute of Technology
Cambridge, MA

AIAA 8th Computational Fluid Dynamics Conference
June 9–11, 1987 / Honolulu, Hawaii

AIAA-87-1167-CP
Honolulu, June 1987

Adaptation Methods for a New Navier-Stokes Algorithm

John G. Kallinderis[*]

Judson R. Baron[†]

Computational Fluid Dynamics Laboratory

Department of Aeronautics and Astronautics

Massachusetts Institute of Technology

Cambridge, MA 02139

Abstract

Various adaptation techniques for the computation of 2-D viscous flows are presented. An initially coarse grid is automatically embedded locally via a feature detection algorithm to provide accurate predictions of boundary-layer regions. The need for resolution in a specific direction can be used to limit embedding to that direction. Lastly,within appreciably viscous regions the full Navier-Stokes equations are solved, while for the remaining areas the description is reduced to the Euler equations. The current procedure combines the three adaptation techniques -viscous, directional, and equation- and attains equivalent accuracy with more than an order of magnitude increase in efficiency over non-adaptive methods.

The basic algorithm uses a new finite volume scheme that has been developed for the discretization of the viscous terms and has the conservation property which is of some importance when shocks are present. Example flowfields that are considered include circular arc cascades in both subsonic and supersonic flow. Comparisons are made with previous results.

INTRODUCTION

In recent years, considerable progress has been made in the development of numerical methods for the solution of the Navier-Stokes equations. Most of those methods however, are not practical for the calculation of complicated flows in a design environment. The primary reason is that the efficiency of the current algorithms is poor and makes it difficult to obtain accurate results.Very fine resolution is needed,which results in long computation times even with the use of available supercomputers. An approach often adopted is to use a simplified set of equations to describe the flow field (e.g. potential flow or Euler equations). Such approximations have proven successful for specific kinds of flow fields, but cannot cope with complicated flows as usually are the flows of engineering interest.

The classical way of achieving the needed resolution is to cluster grid points in regions with high flow gradients. This makes grid generation more difficult (especially in 3-D). More important is that the specific clustering procedure may also create problems related to accuracy and stability due to the resulting grid stretching and skewness. Even for simple geometries, clustering also frequently results in unnecessary resolution in some regions of the domain.

A promising approach is to embed an initially relatively coarse grid locally in regions with large flow-gradients (e.g boundary-layers, shocks, wakes etc). In order to accomplish this the algorithm must sense high gradient regions and automatically must divide the grid-cells in such regions. This approach has been used for the resolution of shocks in flow fields described by Euler equations [1,4,5,6,9,12].

Flow fields involving multiple scale phenomena, such as represented by boundary layers and shocks, are of primary interest here. In order to predict such flows accurately and efficiently, special adaptation techniques as well as a discretization scheme for the viscous terms in the Navier-Stokes system have been developed.

The mentioned inviscid adaptation method has been extended to include viscous regions (viscous adaptation). However, unnecessary and therefore inefficient embedding is avoided by locally refining cells only in the direction of flow gradients (directional adaptation). Similarly, the viscous terms are evaluated only in those regions where vis-

[*]Research Assistant

[†]Professor,Associate Fellow AIAA

cous stresses are appreciable. Elsewhere the description is reduced to the Euler equations (equation adaptation).

In order to correctly model shocks, a conservative discretization scheme is required. The present approach is an extension of an existing finite volume , Lax-Wendroff type scheme [10] with viscous terms added and suitably discretized.

The numerical scheme is presented first and validated by comparison with existing solutions. Then the adaptive techniques are described and computed results will be presented to demonstrate their accuracy. Finally, the efficiency of these techniques is discussed.

GOVERNING EQUATIONS

The Navier-Stokes equations written in cartesian two-dimensional conservation form are :

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = \frac{\partial R}{\partial x} + \frac{\partial S}{\partial y} \tag{1}$$

where

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E \end{pmatrix}, F = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ (E+p)u \end{pmatrix}, G = \begin{pmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ (E+p)v \end{pmatrix} \tag{2}$$

are the state and the convective flux vectors in the x and y-directions respectively. The viscous flux vectors are

$$R = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ u\tau_{xx} + v\tau_{xy} + q_x \end{pmatrix}, S = \begin{pmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ u\tau_{xy} + v\tau_{yy} + q_y \end{pmatrix} \tag{3}$$

where $\tau_{xx}, \tau_{yy}, \tau_{xy}$ are the viscous stresses. For a perfect gas, the pressure is related to the specific internal energy e by $p = (\gamma - 1)\left[e - (\rho/2)\left(u^2 + v^2\right)\right]$

After nondimensionalising the above equations, Mach and Reynolds numbers appear as parameters. Sutherland's law was used for the viscosity coefficient for all computations.

NUMERICAL SCHEME

A previously developed explicit, finite volume numerical scheme which was developed by Ni [10] for the Euler equations was used for the discretisation of the convective terms. The method uses Lax-Wendroff type marching in time and consists of two basic operations. The first of these evaluates the first order temporal terms at the grid nodes (e.g. at E in Figure 1). These terms involve first order spatial derivatives which are evaluated by averaging the surrounding primary control volume center values (e.g. at a through d). The second operation determines the second order temporal terms at the grid nodes. These contributions involve second order spatial derivatives which are calculated using a spatially translated control volume (e.g. abcd in Fig. 1). This operation involves a piecewise integration around this secondary control volume.

Viscous terms

The above Euler scheme has been extended by including the viscous terms of the Navier-Stokes equations. Various other extensions of Euler schemes without adaptation are found in [8](for the Ni-scheme) and [13](for a Runge-Kutta scheme).

The first order temporal viscous terms involve second order derivatives, and the second order temporal terms involve 4th order derivatives. It is important to minimize the number of nodes which contribute to node E, thus only those viscous terms which provide first order temporal accuracy are kept, since our interest is in the steady state. The spatially translated cell *abcd* is used to compute them.

We illustrate the discretisation of the viscous terms by *consideration* of the viscous term u_{xx} at node E. Using Green's theorem for the volume abcd,we have:

$$\begin{aligned} u_{xx} &= (1/S_{abcd}) \oint_{abcd} (u_x) dy \\ &= (1/S_{abcd}) \left[(u_x)_{cd}\Delta y_{cd} + (u_x)_{bc}\Delta y_{bc} + \right. \\ &\qquad \left. (u_x)_{ab}\Delta y_{ab} + (u_x)_{da}\Delta y_{da} \right] \end{aligned}$$

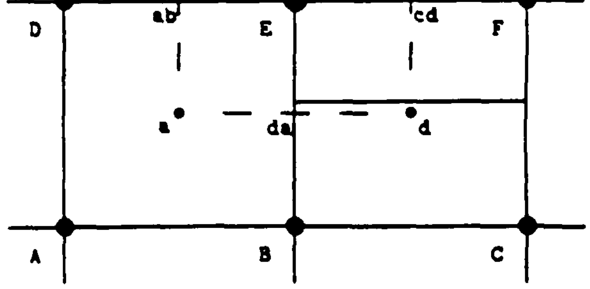


Figure 1: Computational Grid

where S_{abcd} is the area of the cell $abcd$, $\Delta y_{cd} = y_c - y_d$, etc. The first order derivative at the face cd of the control volume $abcd$, is evaluated employing the area $EcFd$. Similar volumes are used for the other face derivatives. Thus,

$$(u_x)_{cd} = (1/S_{cd})[u_F \Delta y_F + u_c \Delta y_c + u_E \Delta y_E + u_d \Delta y_d]$$

where S_{cd} is the area of $EcFd$, and

$$\begin{aligned} \Delta y_F &= (\Delta y_{IF} - \Delta y_{FC})/2 \\ \Delta y_c &= (y_H + y_E)/2 - (y_I + y_F)/2 \\ u_c &= (u_E + u_H + u_I + u_F)/4 \end{aligned}$$

etc. The above discretization is conservative. The discretization of the convective terms allows odd-even modes to appear in both directions but the discretization of the viscous terms does not allow those modes to appear.

In order to accelerate convergence to the steady state a multiple grid method [10] was used which acts only on the convective terms. Its function is to accelerate the propagation of information by using coarser than the basic grids.

Odd-even modes were suppressed in the essentially inviscid portion of the flow with the aid of a nine-point Laplacian smoothing operator of the form

$\mu(U_a + U_b + U_c + U_d - 4U_E)$. For shock capturing the following smoothing operator in conservative form was used

$$\sigma(|\Delta_x p| \Delta_x U + |\Delta_y p| \Delta_y U)$$

where $\Delta_x(.), \Delta_y(.)$ denote differences in the x,y directions. The pressure terms are necessary in order to switch smoothing on and off at and away from the shock regions. Typical values for the smoothing coefficients were: $\mu = 0.002, \sigma = 0.01$

An important property of the above scheme is that all operations can be performed in a piecewise sense within each cell without the need for any information from the outside. This is very useful in dealing with unstructured grids, as will be seen in the following.

Solver Validation

A test of the Navier-Stokes numerical scheme has been carried out for problems for which comparisons with previous results were possible. The first case is a flat plate in a supersonic flow at $M_\infty = 3.$ and $Re = 10^3$(Fig 2) with a 65x65 grid.

Four quantities $(p_0, T_0, v/u, p_\infty)$ were specified at the inlet plane, and a no slip condition, temperature, and extrapolated pressure at the solid surface boundary. At the upper boundary of the domain a tangency condition was applied. All state variables were extrapolated at the supersonic parts of the exit plane, and in the subsonic parts the pressure was fixed at the value in the immediately adjacent supersonic part. In Figure 2 we compare the skin friction distribution at the wall with that from [2].

A second case is a 10% circular arc cascade in a subsonic flow of $M_\infty = 0.5$ and $Re = 8 \times 10^3$ (see Fig 3) with a 65x33 grid. The wall-pressure and C_f curves for this example are compared with [3,8,11] (see figure 3). The agreement is excellent for both cases.

ADAPTIVE TECHNIQUES

Local Grid Refinement

Accuracy is achieved with a minimal amount of computational effort by embedding several levels of finer grids only in those regions of the domain where important features exist.This can be accomplished by simply subdividing cells of the initial coarse grid in both cell-directions (Fig 4). In this way the embedded and initial grids are topologically similar. This means that if the initial grid is uniform and orthogonal, these desirable properties characterise the embedded meshes as well. However, depending on cell and feature orientation,there are situations in which resolution is needed primarily in one direction in the vicinity of the feature. In that case it is advantageous to divide the cell only in that direction and thus avoid the creation of unecessary cells (Fig 4) (directional refinement).

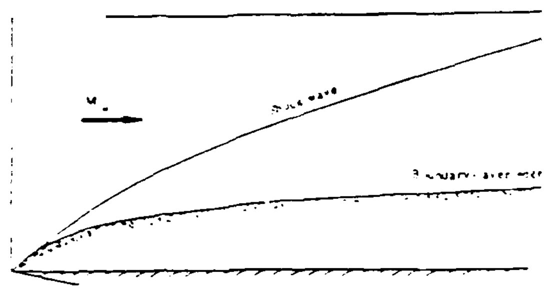
Equation Adaptation

The magnitude of viscous stresses generally decreases very rapidly away from solid boundaries. On the other hand the evaluation of viscous terms is quite expensive. Thus the approach here has been to introduce a criterion to monitor the need for solution of the full Navier-Stokes equations, and to do so only when required by the presence of shear. The Euler equations then are applicable everywhere else. This is a relatively easy procedure with the present explicit algorithm because each cell is integrated independently at each time-step. Essentially different integrators can be used for cells in which different physics dominates.

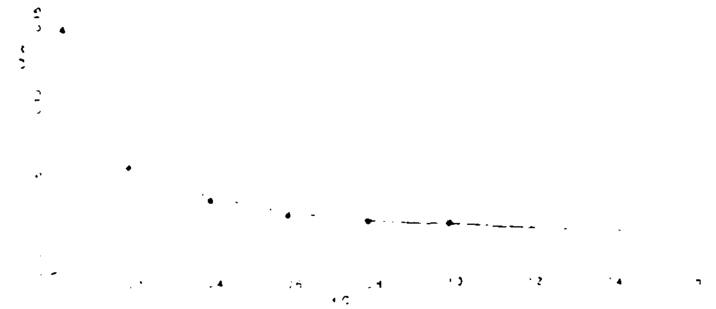
Feature Detection

It is essential in the above adaptation technique that the algorithm be able to sense the existence and track the evolution of special features. The dominant features of interest are shear layers and shocks.

There is a choice of flow parameters that can be used to detect the above features, including velocity, pressure, density and Mach number distributions. Variations which have been examined include undivided and divided differences for these parameters. Shear layers and shocks are very different flow phenomena with completely different scales. Use

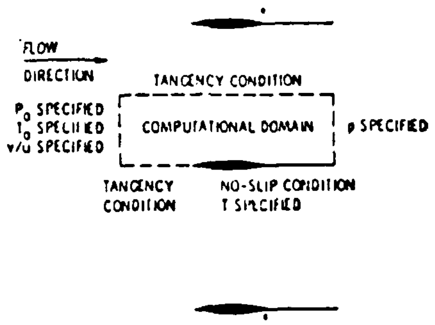


(a) Computational Domain

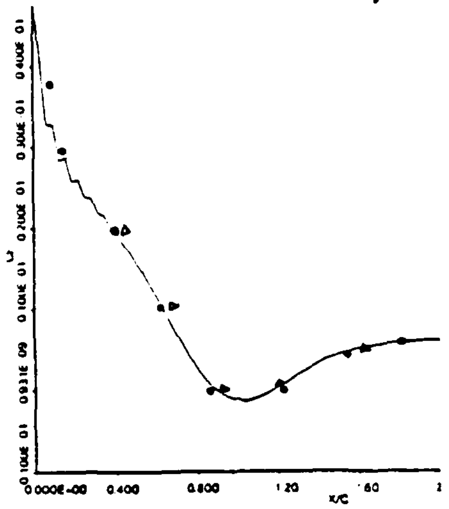


(b) Comparison of skin- friction distributions
— present work
•Carter [2]

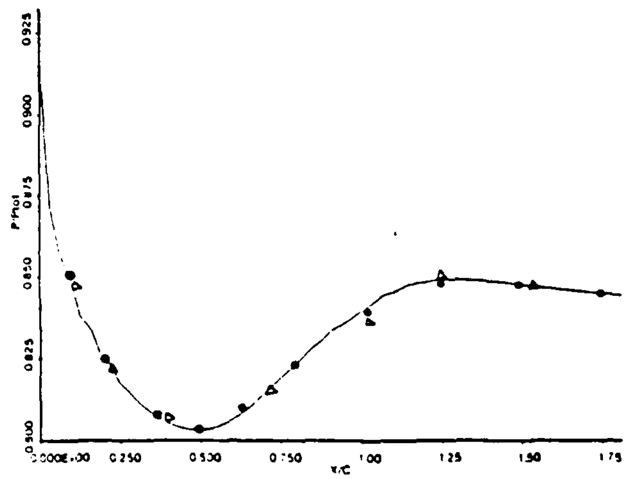
Figure 2: Supersonic flat plate ($M_\infty = 3, Re = 10^3$)



(a) Computational Domain
Boundary Conditions



(b) Comparison of skin- friction distributions
— present work
•Davis [8]
Δ Whitfield [7]



(c) Comparison of wall- pressure distributions
— present work
•Davis [8]
Δ Chima [3]

Figure 3: 10% Circular Arc Cascade ($M_\infty = 0.5, Re = 8 \times 10^3$)

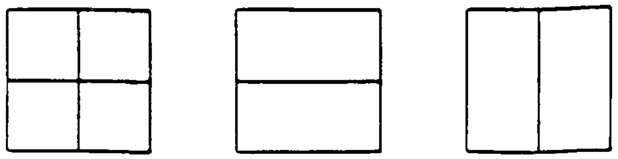


Figure 4: embedding subdivisions

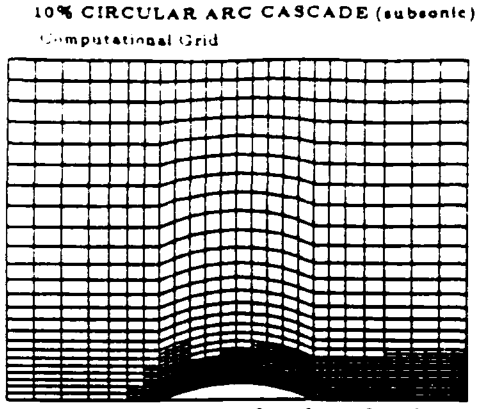
of only one of the above parameters is inadequate. A combination of suitable parameters proves to be necessary.

In Figures 5 and 6 we illustrate the influence of various criteria on the detection of a boundary layer and a shock separately. Fig. 5 illustrates a subsonic flow field with a boundary layer being the main feature , while Fig. 6 shows the same field in supersonic flow. In the latter case, a shock is formed at the leading edge and is reflected at the upper boundary. Both of these flow fields will be described in more detail in the next sections. All the criteria are applied in a directional sense (l,m being the cell-directions). The cells within the boundary layer are generally much smaller than those in the inviscid region. As a consequence, an undivided difference of a detection parameter at an inviscid cell can be of the same magnitude as those at a viscous cell. Therefore divided differences are required for the correct capture of shear layers. On the other hand, the use of parameter gradients when detecting shock regions would lead to increasing gradients after each adaptation and a decision to continue adapting. Thus we have used both divided differences for shear layers and undivided differences for shocks. It is evident that the use of density leads to conservative number of embedded cells. More important is its inappropriatness for incompressible flow. Mach number is a poor parameter for shock detection in the case of weak discontinuities. However, pressure and velocity differences perform quite well in detecting shocks. Our current approach is to use velocity gradients in order to detect the viscous regions and velocity differences in order to detect shocks.

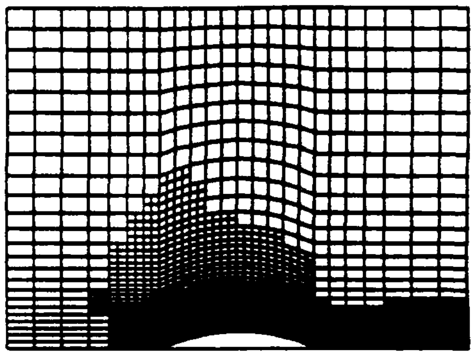
In addition to parameter selection a choice must be made of threshold levels for the detection parameters. The threshold is set by using average and standard deviation values of the parameters. More specifically we use *threshold* = $\Phi_{ave} + \alpha\sigma_{\Phi}$ where $\Phi_{ave}, \sigma_{\Phi}$ are the average and standard deviation values of the detection parameter Φ, and α is a weighting factor chosen empirically but found to be applicable for a variety of flow fields and conditions.

Data Structure

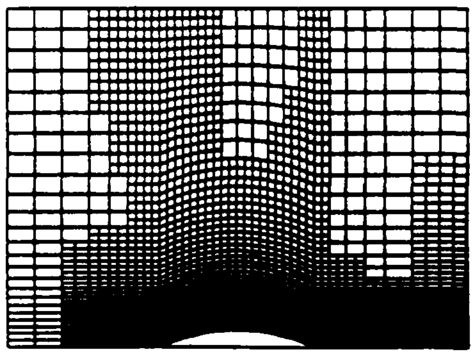
Very important considerations in the adaptive scheme relate to the storage and availability of the information which is necessary for the various calculations. A special data structure is required to service the unstructured grid arising from adaptation. Since each cell is computed independently of its neighbors, the grid-structure has no impact on



(a) velocity gradients ($\frac{\delta_l u}{\delta l}, \frac{\delta_m u}{\delta m}, \frac{\delta_l v}{\delta l}, \frac{\delta_m v}{\delta m}$)

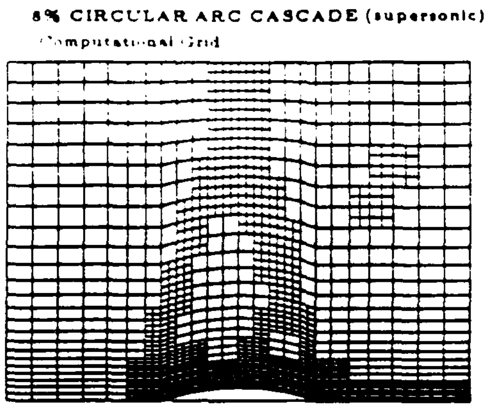


(b) velocity differences ($\delta_l u, \delta_m u, \delta_l v, \delta_m v$)

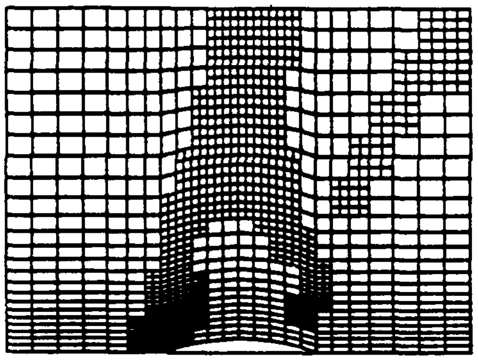


(c) density differences ($\delta_l \rho, \delta_m \rho$)

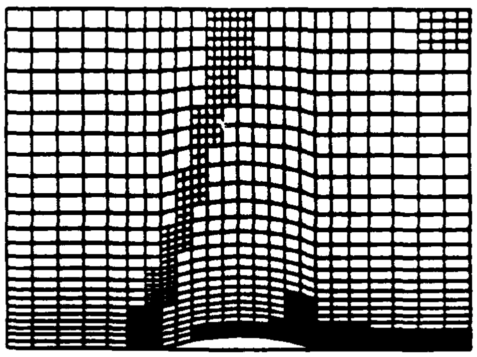
Figure 5: Boundary layer detection



(a) velocity differences ($\delta_l u, \delta_m u, \delta_l v, \delta_m v$)



(b) pressure differences ($\delta_l p, \delta_m p$)



(c) Mach no differences ($\delta_l M, \delta_m M$)

Figure 6: Shock detection

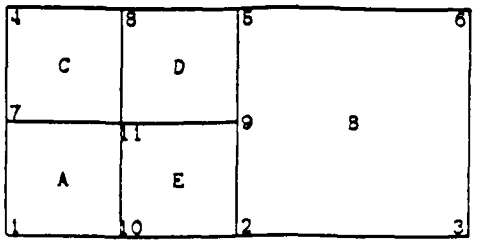


Figure 7: Basic Data Structure

the solver, which remains the same for any grid topology. In particular the pointer system is based here on cells rather than blocks of cells, and this provides great flexibility. A system of cellwise pointers, similar to finite element connectivity arrays, keeps track of the information required by the solver. In Figure 7 we illustrate the basic principle of storing information at the cells of an unstructured grid. The four nodes corresponding to each one of the cells are illustrated (e.g. cell B has the nodes 2, 5, 6, 3; and node 9 is appointed to cells D and E)

Interface Treatment

The existence of embedded regions within the interior of the domain introduces internal boundaries (Fig 8) which must be treated carefully. Stability and accuracy are the two important considerations in the numerical treatment at such an interface. The use of directional refinement introduces several additional types of interfaces and this imposes another requirement on the choice of interface treatment. The treatment preferably should be simple and easily extended to 3-D.

An interface poses problems in the basic solver for two reasons. First is that there are cells containing five nodes, whereas the scheme is designed for cells with nodes at only the four corners. The second is that at the interface there is an abrupt change in the size of the cells. This poses accuracy problems for a Navier-Stokes solver.

One way to approach interface problems, is to perform a special integration for those cells which involve more than four nodes. This is accomplished by modifying the scheme in such a way as to include the additional interface nodes in the integration procedure. It follows that a different integration then would be required for different types of interfaces. This poses a number of problems when using directional embedding as here and especially for 3-D fields, because many different types of interfaces appear.

Our approach has been to treat all possible interface configurations in a unique way. This is accomplished by disregarding 'problematic' interface cells and using other existing cells of the domain instead in order to perform the calculations at the interfaces. More specifically let us examine one type of interface at Figure 8. The nodes a, b are integrated

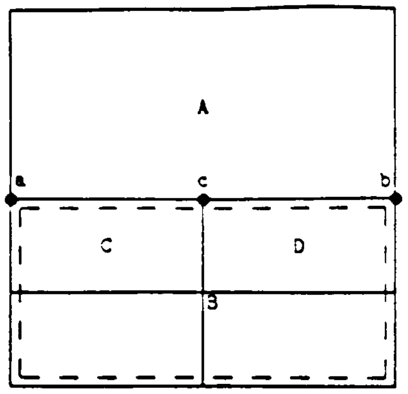


Figure 8: Interface Treatment

using the parent cell B instead of cells C and D, and ignoring the center node c. Thereafter the values at node c follow by interpolating from nodes a,b. The same approach is employed for all kinds of interfaces. This treatment is easily extended to 3-D and has proven to be accurate and robust in the cases considered so far, even when a shock intersects an interface.

Adaptive Solution Procedure

The solution algorithm that was used consists of the following steps:

1. Initialise the field with a uniform, orthogonal coarse grid on which the Navier -Stokes description is applied.
2. Monitor the residual until it falls below a prespecified value; detect the main flow features; refine the grid locally
3. Continue the computation on the updated grid using Euler/Navier-Stokes solvers for the inviscid/viscous cells.
4. Repeat steps 2,3 for a specified number of cycles.
5. March the solution to steady state

At each embedding level the error wave-lengths corresponding to the cell-sises of that level are smoothed. In that sense, adaptation plays the role of a multigrid method.

Coupling of Grids

Two distinct types of cells are involved during the solution procedure. There are the locally finest cells that are used by the basic solver to integrate the equations, and the multiple grid coarser cells which are used by the multiple grid algorithm after being created from the fine cells by deleting

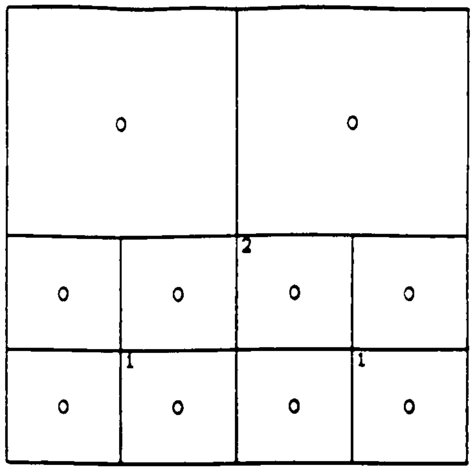


Figure 9: Grids' Coupling

every other grid line (Fig. 9). The locally finest cells may belong to either an unembedded or an embedded region (see Fig. 9).

The solution procedure starts from a sweep of the locally finest (0-level) cells. Then the multiple grid accelerator is used within the embedded regions (multiple grid level 1 in Fig. 9). Finally the multiple grid operator is used throughout the whole domain (level 2 in fig. 9). Notice that the interfaces are 'invisible' after completing the multigrid levels within the embedded regions (level 1 in fig. 9).

RESULTS

Model problems with flow past a circular arc cascade in both subsonic and supersonic flow have been used to evaluate the accuracy and efficiency of the adaptation techniques.

Subsonic Flow

The flow at $M_\infty = 0.5$ and $Re = 8 \times 10^3$ was calculated using an initial 25x25 mesh with uniform spacing across the boundary layer and up to three levels of embedding allowed in both directions. The final grid is shown in Figure 10. The area near the trailing edge proves to need less embedding because the boundary layer separates and the fluid is virtually stagnant with negligible stresses . The three peaks in the convergence history (Figure 10) mark the upset introduced by the adaptive embedding of the grid. It is to be noted that adaptation does not alter the overall slope of the curve.

Indeed the presence of interfaces in the flow field does not affect the solution. This is illustrated in Figure 11. In the former we see the separated velocity profile at the trailing edge station with three marks indicating the position of the interfaces. In the latter the density contour plots are shown, and in neither are kinks observed due to the interfaces. Of cource, greater resolution is needed across the boundary layer than in the streamwise direction. Applying directional embedding across the boundary layer at the 3rd level results in significant savings in the number of cells, with no apparent change in the results on comparing wall shear distributions obtained using adaptation in either both directions or directionally (Figure 12).

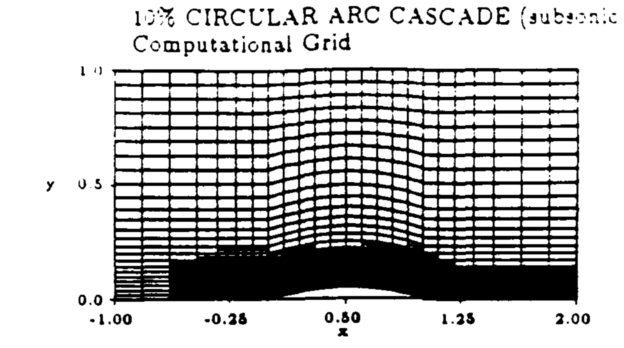
Supersonic Flow

An 8% circular arc cascade,with $M_\infty = 1.4$ and $Re = 23 \times 10^3$ again with a 25x25 initial mesh was used with uniform spacing across the boundary layer and an allowance for three levels of embedding in both cell-directions. An oblique shock forms at the leading edge as expected and is reflected at the upper symmetry boundary. The reflected shock then interacts with the boundary layer at the trailing edge region. In Figure 13 , the grid evolution is illustrated and demonstrates how the embedded grids follow the detailed physics of the flow. Figure 13 (e) provides an enlarged detail of the grid near the surface. The boundary layer is essentially 'lifted' by the adverse pressure gradient which is induced by the reflected shock. Simultaneously, because of the effective corner which is formed by the boundary layer, compression waves are formed upstream of the interaction region and coalesce into a weaker shock which impinges on the upper boundary. Note that passage of the shock through interfaces does not induce any stability problems. Further flow details are shown in the Mach number and pressure coefficient, C_p contour plots in Figure 14 . Note that the slip line due to the Mach reflection at the upper boundary is clearly observed in the Mach number contour plot(Figure 14). Notice also the deflection of the reflected shock at the shock/boundary layer interaction region, and the separated recirculating region at the trailing edge. Expansion fans are formed at the interaction region because of the effective wedge formed by the separated boundary layer. The local embedding procedure appears to be effective in capturing the detailed physics of a flow field in the presence of rather complicated multiple -scale phenomena.

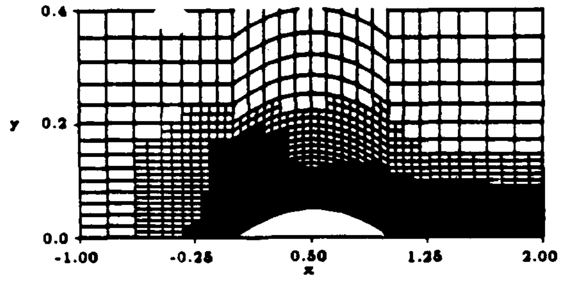
In order to evaluate the accuracy in this case, a globally fine grid (97x97) corresponding to two level embedding was employed and the results were compared with those obtained by using two levels of local, adaptive embedding. Figure 15 compares the wall- pressure and wall-shear distributions. The agreement proves to be excellent.

ADAPTATION EFFICIENCY

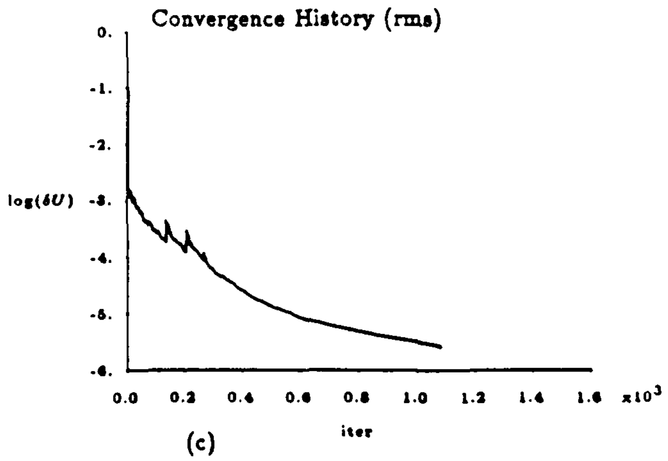
The fact that the entire Navier-Stokes system is solved only within the viscous regions leads to a 20% time savings due solely to the equation adaptation scheme.



(a) original scales

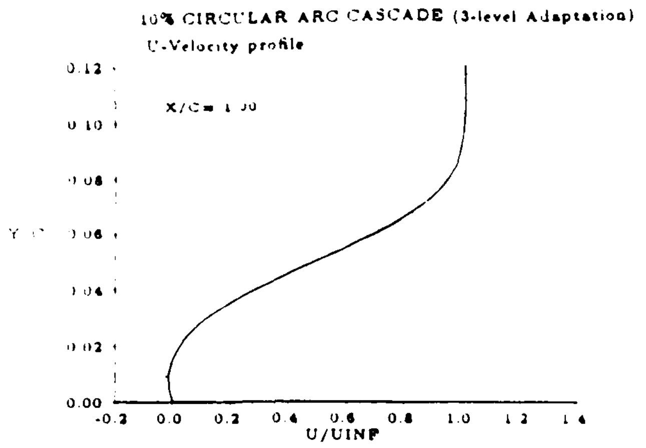


(b) enlarged scale

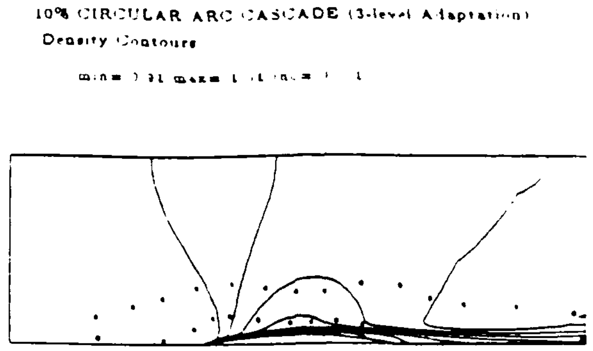


(c)

Figure 10: Embedding (3-levels)

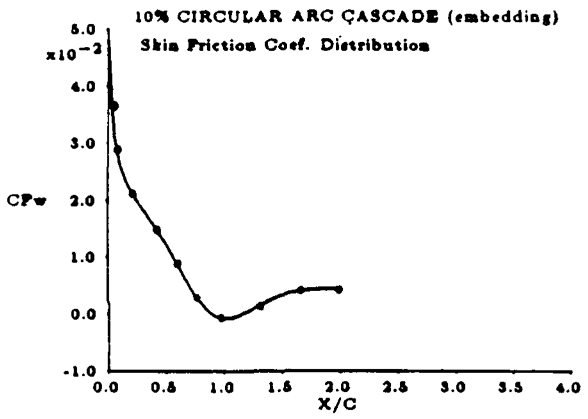


(a) : interface position



(b) • interface position

Figure 11: **Effect of Interfaces**



— embedding in both directions
•3rd embedding level in one dir.

Figure 12: Comparison of wall-shear

The directional embedding which was applied in the subsonic case at the 3rd level led to a 40% decrease in the number of cells compared to the cells formed if embedding in both directions was applied. This is not surprising in view of the fact that a cell divided in both directions adds three additional cells, while directional division adds only one additional cell. This results in significant savings in the number of cells, and especially so at the higher adaptation levels when many new cells are created. Directional embedding should be even more beneficial in turbulent boundary layers where at least two nodes are needed inside the laminar sublayer.

The subsonic case on a globally fine mesh would require 140 hours on a MicroVax computer, making the conservative assumption that the same number of iterations would be required as for the embedded case. With adaptation in both cell-directions, only 11 hours are required, and this reduces to 6 hours with allowance for directional embedding at the 3rd level. Finally, 5 hours are required if equation adaptation is also applied. Overall, the adaptive techniques lead to a 28 times reduction in CPU time. The gain would be appreciably larger if more than three adaptation levels were permitted.

In the supersonic case,the reduction factor is 22. The slightly smaller advantage follows from the fact that there are more flow features to resolve, and these result in a larger number of embedded cells.

CONCLUSIONS

A conservative finite volume discretization of the viscous terms of the Navier-Stokes equations has been developed using the same stencil as for the convective terms.

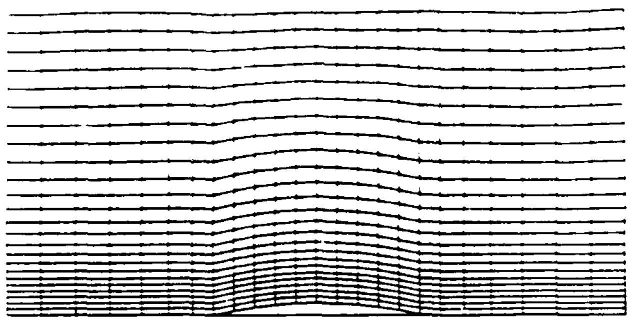
Grid and equation adaptation have been carried out for multiple and overlapping flow features.

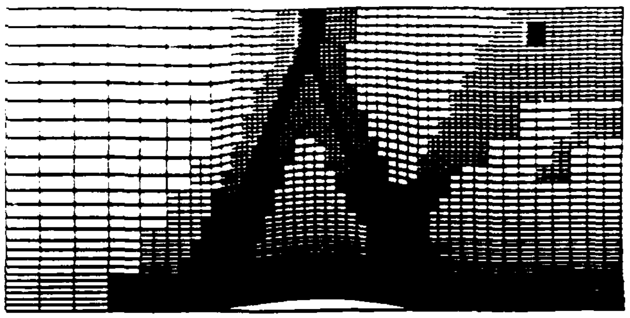
The combined use of viscous, directional and equation adaptation has indicated CPU time reduction factors of approximately 25 for subsonic and supersonic cascade examples for equivalent accuracies.

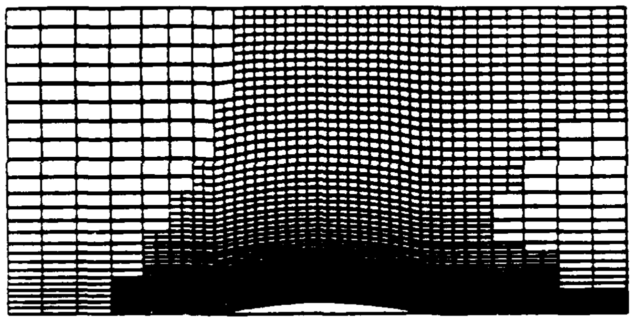
ACKNOWLEDGEMENTS

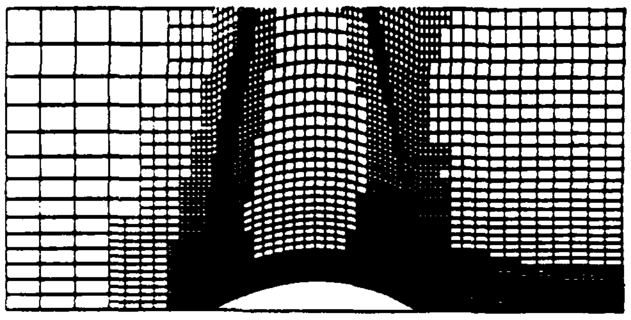
This work was supported by the Air Force office of Scientific Research under grant AFOSR-82-0136, Dr J.D.Wilson technical monitor. The authors would also like to thank Prof. M. Giles for his suggestions and discussions.

5% Circular Arc Cascade (supersonic)
computational grid


(a) initial grid (25x25)


(d) third embedding level


(b) first embedding level


(e) third embedding level
(vertical scale enlarged)

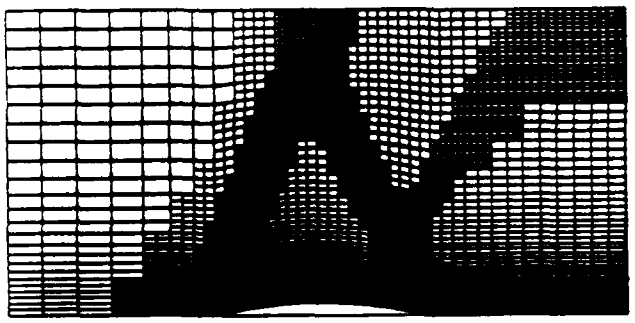

(c) second embedding level

Figure 13: **Grid evolution**

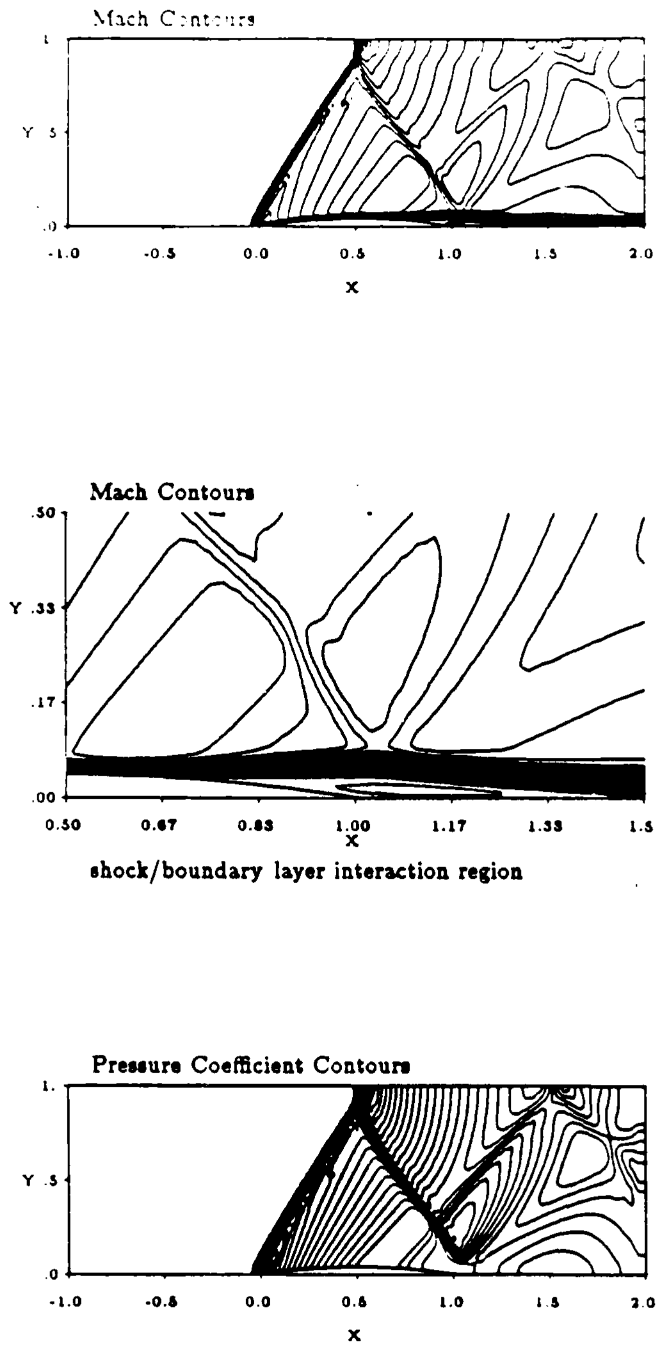


Figure 14: 8% circular arc cascade (supersonic)

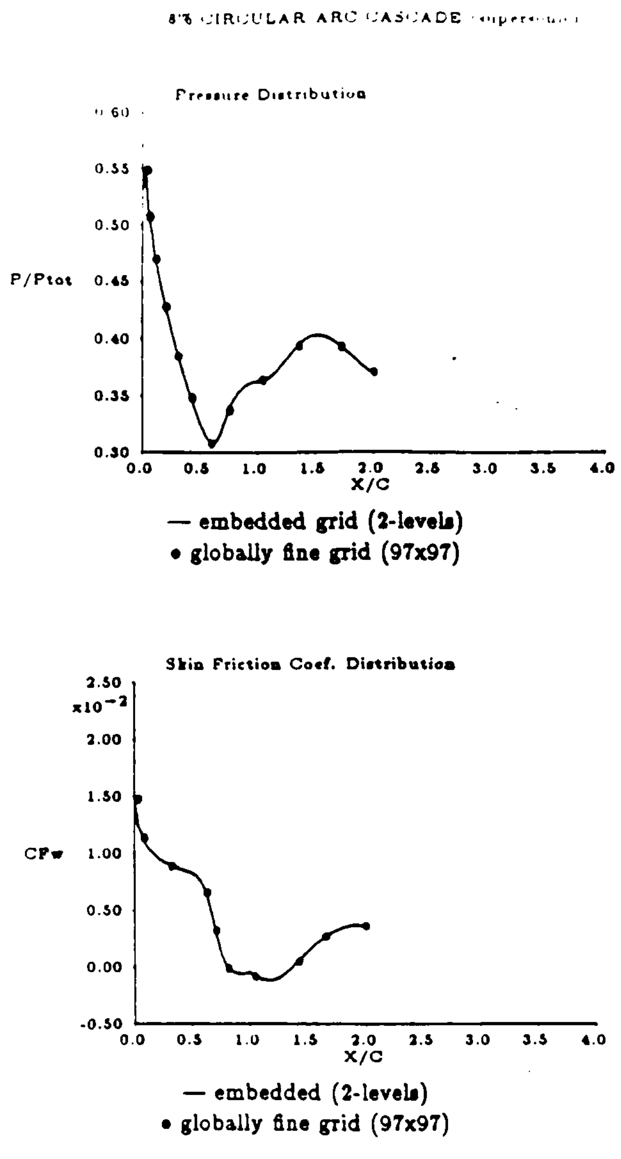


Figure 15: **Adaptation accuracy**

References

[1] M. Berger and A. Jameson. Automatic Adaptive Grid Refinement for the Euler Equations. *AIAA Journal*, 23:561–568, Apr 1985.

[2] J. E. Carter. *Numerical Solutions of the Navier-Stokes Equations for the Supersonic Laminar Flow Over A 2-D Compression Corner.* Technical Report NASA TR R-385, NASA,Langley Research Center, 1972.

[3] R. V. Chima. Efficient Solution of theEuler and Navier-Stokes Equations with a Vectorized Multiple-Grid Algorithm. *AIAA Journal*, 23:23–32, Jan 1985.

[4] J. F. Dannenhoffer III and J. R. Baron. *Adaptive Procedure for Steady State Solution of Hyperbolic Equations.* AIAA Paper 84-0005, 1984.

[5] J. F. Dannenhoffer III and J. R. Baron. *Grid Adaptation for the 2-D Euler Equations.* AIAA Paper 85-0484, 1985.

[6] J. F. Dannenhoffer III and J. R. Baron. *Robust Grid Adaptation for Complex Transonic Flows.* AIAA Paper 86-0495, 1986.

[7] R. L. Davis. Personal communication, 1986.

[8] R. L. Davis, Ron-Ho Ni, and J.E. Carter. *Cascade Viscous Flow Analysis Using the Navier-Stokes Equations.* AIAA Paper 86-0033, 1986.

[9] R. Lohner, K. Morgan, J. Peraire, and O. C. Zienkiewicz. *Finite Element Methods for High Speed Flows.* AIAA Paper 85-1531, 1985.

[10] R.-H. Ni. A Multiple Grid Scheme for Solving the Euler Equations. *AIAA Journal*, 20:1565–1571, Nov 1981.

[11] C. M. Rhie. *A Pressure Based Navier-Stokes Solver Using the Multigrid Method.* AIAA Paper 86-0207, 1986.

[12] R. A. Shapiro and E. M. Murman. *Cartesian Grid Finite Element Solutions to the Euler Equations.* AIAA Paper 87-0559, 1987.

[13] R. C. Swanson and E. Turkel. *A Multistage Time-Stepping Scheme for the Navier-Stokes Equations.* AIAA Paper 85-0035, 1985.

Paper presented at the 7th GAMM Conference on Numerical Methods in Fluid Mechanics, Louvain-la-Neuve, Belgium, Sept. 11, 1987.

A SEMI-IMPLICIT SCHEME FOR THE NAVIER-STOKES EQUATIONS

Bernard Loyd
Earll M. Murman
Saul S. Abarbanel

Computational Fluid Dynamics Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139, USA

Abstract

This paper presents a novel *Semi-Implicit Navier Stokes Solver (SINSS)*. *SINSS* combines the advantages of implicit with those of explicit schemes: temporal integration is implicit in the direction normal to a body and explicit in the direction(s) tangential to it. Numerical stiffness due to disparate physical scales in the normal direction is eliminated, and the stability of the algorithm depends only on relatively coarse streamwise grid spacing – not on the typically fine normal spacing. Approximate factorisation is unnecessary and only one matrix inversion per stage is required.

The semi-implicit solver is applied to a finite volume formulation of the $2-D$ thin layer Navier-Stokes equations. Efficiency of the algorithm is studied by comparison of convergence histories of the semi-implicit algorithm with those of a multigrid explicit scheme and a fully implicit approximately factorised scheme. The effect of residual smoothing is also considered. Computations show that the explicit, semi-implicit, and fully implicit schemes are of comparable efficiency for inviscid calculations. *SINSS* is superior in high Reynolds number flows, where multigrid loses effectiveness and the implicit scheme appears to have convergence problems.

1 Introduction

Fine spatial resolution for Navier-Stokes simulations is often necessary only in the direction normal to a body. At the differential equation level this implies that only the boundary layer like viscous terms need to be retained, leading to the so called thin layer Navier-Stokes equations. At the discrete level this implies mesh cells will have a much smaller dimension in the normal direction compared to the streamwise direction. Thus, the stability restriction for explicit schemes in body fitted grid systems is usually dominated by the small normal spacing and results in numerical "stiffness" in the equations.

The *SINSS* solver eliminates the stability restriction due to the normal spacing by solving the flow equations implicitly in the normal direction. However, the solver is explicit in the tangential (flow) direction, thereby avoiding factorisation (AF) schemes, *CFL* limitations associated with the AF error, and a second or second and third block-tridiagonal inversions in two or three dimensions. This scheme was first presented by [Loyd et al. 86]. Details may be found in [Loyd et al. 87].

This paper presents the characteristics and efficiency of a semi-implicit algorithm as applied to a popular multi-stage scheme. First, we present the Navier-Stokes equations, and a simple method of implementing the thin layer approximation that preserves the conceptual simplicity and conservative property of the finite volume approach. The semi-implicit algorithm is then derived by considering time linearisation of only the cross flow flux terms. The *CFL* limit on the time step for explicit schemes reduces to a one-dimensional *CFL* restriction based only upon the streamwise terms [Loyd et al. 87].

To evaluate the relative efficiency of the semi-implicit we compare convergence histories obtained with it to those obtained with an explicit and a fully implicit solver [Beam & Warming 76]. Multigrid (MG) and residual smoothing (RS) are applied in the explicit solver to accelerate convergence, and RS is also applied in the streamwise (explicit) direction in the semi-implicit scheme. The paper is concluded with results for inviscid and viscous flow cases.

2 2–D Navier-Stokes Equations

The two-dimensional Navier–Stokes equations integrated in Cartesian coordinates over a control surface Ω with boundary $\partial\Omega$ are:

$$\frac{\partial}{\partial t}\iint_{\Omega} \mathbf{W}\, dS + \oint_{\partial\Omega} (\mathbf{F}\, dy - \mathbf{G}\, dx) = 0. \tag{1}$$

$\mathbf{W} = (\rho\ \rho u\ \rho v\ E)^T$ is the vector of state variables, where ρ, u, v are density, x and y components of velocity, and E is total energy. The flux vectors $\mathbf{F}$ and $\mathbf{G}$ are

$$\mathbf{F} = \begin{pmatrix} \rho u \\ \rho u^2 + P + \tau_{xx} \\ \rho uv + \tau_{xy} \\ \rho uH + u\tau_{xx} + v\tau_{xy} - q_x \end{pmatrix}, \qquad \mathbf{G} = \begin{pmatrix} \rho v \\ \rho uv + \tau_{yx} \\ \rho v^2 + P + \tau_{yy} \\ \rho vH + u\tau_{yx} + v\tau_{yy} - q_y \end{pmatrix}. \tag{2}$$

H is the stagnation enthalpy ($H = E + P/\rho$), and the viscous stresses are, with the Stokes hypothesis:

$$\begin{aligned} \tau_{xx} &= \tfrac{2}{3}\mu\left(2\tfrac{\partial u}{\partial x} - \tfrac{\partial v}{\partial y}\right) & \tau_{yy} &= \tfrac{2}{3}\mu\left(2\tfrac{\partial v}{\partial y} - \tfrac{\partial u}{\partial x}\right) \\ \tau_{xy} &= \tau_{yx} = \mu\left(\tfrac{\partial u}{\partial y} + \tfrac{\partial v}{\partial x}\right) \\ q_x &= \tfrac{\mu}{Pr}\left(\tfrac{\partial H}{\partial x} - u\tfrac{\partial u}{\partial x} - v\tfrac{\partial v}{\partial x}\right) & q_y &= \tfrac{\mu}{Pr}\left(\tfrac{\partial H}{\partial y} - u\tfrac{\partial u}{\partial y} - v\tfrac{\partial v}{\partial y}\right). \end{aligned} \tag{3}$$

Pr is the Prandtl number, $Pr = \frac{\mu C_p}{k}$. The equation of state for a perfect gas closes the system:

$$P = (\gamma - 1)\left[E - \frac{1}{2}\rho(u^2 + v^2)\right]. \tag{4}$$

2.1 Thin Layer Approximation

The thin layer approximation is appropriate for high Reynolds number flow cases where shear stresses in the body normal direction are much larger than other viscous stresses. We restrict our attention to flows where this is the case and apply the thin layer approximation.

Because of the assumptions in the thin layer form of the Navier Stokes equations, the computational (ξ, η) grid must contain a family of lines that is body normal or nearly so. Since the normal direction η in general does not correspond to either the x or y coordinate direction, the derivatives in the viscous stresses must be transformed to ξ and η coordinates via the generalised transformation:

$$\frac{\partial}{\partial x} = \frac{\partial \xi}{\partial x}\frac{\partial}{\partial \xi} + \frac{\partial \eta}{\partial x}\frac{\partial}{\partial \eta} = \xi_x\frac{\partial}{\partial \xi} + \eta_x\frac{\partial}{\partial \eta} \qquad \frac{\partial}{\partial y} = \frac{\partial \xi}{\partial y}\frac{\partial}{\partial \xi} + \frac{\partial \eta}{\partial y}\frac{\partial}{\partial \eta} = \xi_y\frac{\partial}{\partial \xi} + \eta_y\frac{\partial}{\partial \eta}. \tag{5}$$

Using the assumption $\frac{\partial}{\partial \eta} >> \frac{\partial}{\partial \xi}$, all ξ derivative terms may be dropped to obtain:

$$\frac{\partial}{\partial x} = \eta_x\frac{\partial}{\partial \eta}, \qquad \frac{\partial}{\partial y} = \eta_y\frac{\partial}{\partial \eta}. \tag{6}$$

The thin layer shear stresses and heat fluxes become:

$$\begin{aligned} \tau_{xx} &= \tfrac{2}{3}\mu\left(2\eta_x\tfrac{\partial u}{\partial \eta} - \eta_y\tfrac{\partial v}{\partial \eta}\right) & \tau_{yy} &= \tfrac{2}{3}\mu\left(2\eta_y\tfrac{\partial v}{\partial \eta} - \eta_x\tfrac{\partial u}{\partial \eta}\right) \\ \tau_{xy} &= \tau_{yx} = \mu\left(\eta_y\tfrac{\partial u}{\partial \eta} + \eta_x\tfrac{\partial v}{\partial \eta}\right) \\ q_x &= \tfrac{\mu}{Pr}\left(\eta_x\tfrac{\partial H}{\partial \eta} - u\eta_x\tfrac{\partial u}{\partial \eta} - v\eta_x\tfrac{\partial v}{\partial \eta}\right) & q_y &= \tfrac{\mu}{Pr}\left(\eta_y\tfrac{\partial H}{\partial \eta} - u\eta_y\tfrac{\partial u}{\partial \eta} - v\eta_y\tfrac{\partial v}{\partial \eta}\right) \end{aligned} \tag{7}$$

The viscous terms need only be computed at faces 1 and 3 (Figure 1). The viscous fluxes across faces 2 and 4 are discarded since, by assumption, they are small in comparison.

3 Spatial Discretization

The governing equations must be discretised before numerical solution can be attempted. Discretisation may proceed in two steps, spatial and temporal. Although the two steps are not independent, since stability of the temporal integration depends on the form of the spatial operator, it is convenient to consider them separately. In this section we discuss the spatial discretisation, as well as the related topics of boundary conditions and artificial viscosity.

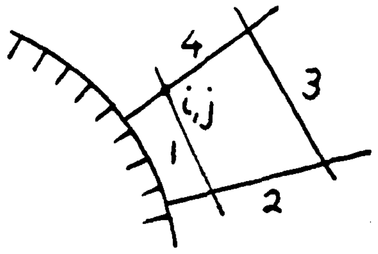


Figure 1: Body Normal Geometry

3.1 Spatial Difference Operators

The line integrals in (1) are replaced by a discrete summation of fluxes across each cell face:

$$\oint_{\partial\Omega} (\mathbf{F}\,dy - \mathbf{G}\,dx) \cong \sum_{l=1}^{4} \mathbf{F}_l \Delta y_l - \mathbf{G}_l \Delta x_l \tag{8}$$

$\mathbf{F}_l$ and $\mathbf{G}_l$ are approximated by simple averages of the flux vectors at the adjacent cells, and, at *faces* 3 and 1, a contribution from the viscous terms:

$$\begin{aligned} \mathbf{F}_2 &= .5 \times (\mathbf{F}_{i,j} + \mathbf{F}_{i+1,j}) \\ \mathbf{F}_3 &= .5 \times (\mathbf{F}_{i,j} + \mathbf{F}_{i,j+1}) + \mathbf{F}_{vis}\,. \end{aligned} \tag{9}$$

Δ's indicate differences taken in the counterclockwise direction, e.g.:

$$\begin{aligned} \Delta y_2 &= y_{i+1,j+1} - y_{i+1,j} \\ \Delta y_3 &= y_{i,j+1} - y_{i+1,j+1}\,. \end{aligned} \tag{10}$$

The viscous operator is written in finite difference form as either

$$\frac{\partial(\,)}{\partial\eta} = \frac{(\,)_{j+1} - (\,)_j}{\Delta\eta} \quad \text{or} \quad \frac{\partial(\,)}{\partial\eta} = \frac{(\,)_j - (\,)_{j-1}}{\Delta\eta}\,, \tag{11}$$

depending on whether the viscous flux at *face 3* or at *face 1* is desired. The appropriate metric η_x or η_y is obtained from the definitions:

$$\eta_x = -\frac{1}{J}\frac{\partial y}{\partial\xi} \qquad \eta_y = \frac{1}{J}\frac{\partial x}{\partial\xi}\,, \tag{12}$$

where J is the Jacobian $J = x_\xi y_\eta - x_\eta y_\xi$. The metrics defining J are calculated with centered differences.

The discretization reduces to a second order accurate centered difference approximation on a Cartesian grid. That order of accuracy is not maintained on arbitrary grids, and it is important that grids vary smoothly to limit degradation in accuracy. It is generally true that for a given order of accuracy a compact stencil such as this will result in a smaller truncation error than a less compact stencil.

Upon forming the viscous terms at each face, the finite volume line integration proceeds as before. The definitions (11) are convenient because they require only a three point stencil for the thin layer Navier Stokes terms. Because it is written in finite volume form the scheme identically conserves mass, momentum, and energy.

3.2 Artificial Viscosity

The discretization described above allows odd-even decoupling of state vector values at adjacent points. Due to aliasing errors, this decoupling may inhibit convergence of the temporal integration. Also, in inviscid regions one finds that discontinuities in the flow field may cause divergence of the algorithm. Artificial viscosity is added to the physical fluxes to damp out the non-physical odd-even oscillations and stabilise the integration in areas of discontinuos flows such as found around a shock wave. We use a pressure weighted blend of second and fourth differences that has been established as particularly effective [Jameson et al. 81]. In regions of viscous flow the artificial viscosity is turned off.

3.3 Boundary Conditions

Numerical boundary conditions must be imposed at the body and at inflow and outflow. For viscous cases we set the velocity at the body equal to sero and maintain only viscous and pressure contributions to the flux integrals (8). Riemann invarient boundary conditions are specified at the inflow, and, for inviscid cases, at the outflow. For viscous flows, all quantities except the pressure, which is set from the freestream, are extrapolated from the interior at the outflow boundary.

4 Semi-Implicit Approach

This section describes the temporal discretisation of the governing equations. We begin with a synopsis of an explicit multistage approach, which also serves to introduce the nomenclature. More importantly, the semi-implicit approach presented in the following section is easily derived via consideration of the explicit integration.

Various techniques for accelerating convergence are often used with explicit schemes. To enable a fair comparison we implement residual smoothing and multigrid in the explicit scheme in Sections 4.4 and 4.5. We also apply residual smoothing in the explicit direction in *SINNS.*

4.1 Explicit Multi-Stage Integration

A popular multistage scheme for fluid dynamic calculations [Jameson et al. 81] is given as:

$$\begin{aligned}
\mathbf{W}^0 &= \mathbf{W}^n \\
\mathbf{W}^1 &= \mathbf{W}^0 - \alpha_1 \tfrac{\Delta t}{S} \left[\textstyle\sum_{l=1}^4 \mathbf{F}_l^0 \Delta y_l - \mathbf{G}_l^0 \Delta x_l - \mathbf{D}^0\right] \\
\mathbf{W}^2 &= \mathbf{W}^0 - \alpha_2 \tfrac{\Delta t}{S} \left[\textstyle\sum_{l=1}^4 \mathbf{F}_l^1 \Delta y_l - \mathbf{G}_l^1 \Delta x_l - \mathbf{D}^0\right] \\
\mathbf{W}^3 &= \mathbf{W}^0 - \alpha_3 \tfrac{\Delta t}{S} \left[\textstyle\sum_{l=1}^4 \mathbf{F}_l^2 \Delta y_l - \mathbf{G}_l^2 \Delta x_l - \mathbf{D}^0\right] \\
\mathbf{W}^4 &= \mathbf{W}^0 - \quad \tfrac{\Delta t}{S} \left[\textstyle\sum_{l=1}^4 \mathbf{F}_l^3 \Delta y_l - \mathbf{G}_l^3 \Delta x_l - \mathbf{D}^0\right] \\
\mathbf{W}^{n+1} &= \mathbf{W}^4 \,.
\end{aligned} \tag{13}$$

Superscripts denote temporal stages. The artificial viscosity, operator $\mathbf{D}$ is frozen at the first stage to minimise computational effort. The constants $(\alpha_1, \alpha_2, \alpha_3)$ are equal $(1/4, 1/3, 1/5)$, except in the multigrid scheme. Vectors are in bold print. Matrices will be denoted by [ ].

The time step Δt that may be taken is limited by the *CFL* condition. The four stage scheme above, with

$$\Delta t = \lambda \frac{S}{|\mathbf{V} \cdot d\mathbf{L}|_{max} + C|d\mathbf{L}_{max}|}\,, \tag{14}$$

where $\mathbf{V} = u\vec{i} + v\vec{j}$, C is speed of sound, and $d\mathbf{L} = \Delta x\vec{i} + \Delta y\vec{j}$, is stable for $\lambda \leq 2\sqrt{2}$.

4.2 Semi-Implicit Integration

The explicit temporal time stepping (13) can easily be converted to a semi-implicit discretisation. Consider the first stage of a multistage scheme with implicit semi-discretisation of the normal component of the flux vectors:

$$\begin{aligned}
\mathbf{W}^1 - \mathbf{W}^0 = -\alpha_1 \tfrac{\Delta t}{S} \quad &[(\mathbf{F}^1 \Delta y - \mathbf{G}^1 \Delta x)_1 + (\mathbf{F}^0 \Delta y - \mathbf{G}^0 \Delta x)_2 + \\
&(\mathbf{F}^1 \Delta y - \mathbf{G}^1 \Delta x)_3 + (\mathbf{F}^0 \Delta y - \mathbf{G}^0 \Delta x)_4 - \mathbf{D}^0] \,.
\end{aligned} \tag{15}$$

Each term in (15) gives the flux across one of the grid faces. Using the standard Newton type linearisation for $\mathbf{F}$ and $\mathbf{G}$, we write,

$$\begin{aligned}
\mathbf{F}^1 &= \mathbf{F}^0 + \frac{\partial \mathbf{F}^0}{\partial t}\Delta t + O(\Delta t^2) & \mathbf{G}^1 &= \mathbf{G}^0 + \frac{\partial \mathbf{G}^0}{\partial t}\Delta t + O(\Delta t^2) \\
&= \mathbf{F}^0 + [A]\Delta \mathbf{W} + O(\Delta t^2) & &= \mathbf{G}^0 + [B]\Delta \mathbf{W} + O(\Delta t^2)
\end{aligned} \tag{16}$$

where $[A]$ and $[B]$ are defined as the 4×4 Jacobian matrices $[\partial\mathbf{F}^0/\partial\mathbf{W}^0]$ and $[\partial\mathbf{G}^0/\partial\mathbf{W}^0]$, respectively, and $\Delta\mathbf{W}^1 \equiv \mathbf{W}^1 - \mathbf{W}^0$. Inserting (16) into (15) and reordering gives

$$\left[[I] + \alpha_1\frac{\Delta t}{S}\left([A]_1\Delta y_1 + [A]_3\Delta y_3 - [B]_1\Delta x_1 - [B]_3\Delta x_3\right)\right]\Delta\mathbf{W} = -\alpha_1\frac{\Delta t}{S}\left[\sum_{l=1}^{4}\mathbf{F}_l\Delta y_l - \mathbf{G}_l\Delta x_l - \mathbf{D}\right]^0 . \tag{17}$$

This is an implicit matrix equation for $\Delta\mathbf{W}$. The *RHS* is the usual semi-discrete form of the residual, while the *LHS* differs from $[I]$ due to introduction of the terms from the time linearisation of $\mathbf{F}$ and $\mathbf{G}$. Because the *LHS* contains only dependent variables at $(j-1, j, j+1)$, it is a block tridiagonal system of equations.

Subsequent steps in a multistage scheme have the same form. For a four step scheme:

$$\begin{aligned}
\mathbf{W}^0 &= \mathbf{W}^n \\
[\mathrm{LHS}]^0\,\Delta\mathbf{W}^1 &= -\alpha_1\tfrac{\Delta t}{S}\left[\textstyle\sum_{l=1}^{4}\mathbf{F}_l^0\Delta y_l - \mathbf{G}_l^0\Delta x_l - \mathbf{D}^0\right] \\
[\mathrm{LHS}]^1\,\Delta\mathbf{W}^2 &= -\alpha_2\tfrac{\Delta t}{S}\left[\textstyle\sum_{l=1}^{4}\mathbf{F}_l^1\Delta y_l - \mathbf{G}_l^1\Delta x_l - \mathbf{D}^0\right] - \Delta\mathbf{W}^1 \\
[\mathrm{LHS}]^2\,\Delta\mathbf{W}^3 &= -\alpha_3\tfrac{\Delta t}{S}\left[\textstyle\sum_{l=1}^{4}\mathbf{F}_l^2\Delta y_l - \mathbf{G}_l^2\Delta x_l - \mathbf{D}^0\right] - (\Delta\mathbf{W}^2 + \Delta\mathbf{W}^1) \\
[\mathrm{LHS}]^3\,\Delta\mathbf{W}^4 &= -\ \ \tfrac{\Delta t}{S}\left[\textstyle\sum_{l=1}^{4}\mathbf{F}_l^3\Delta y_l - \mathbf{G}_l^3\Delta x_l - \mathbf{D}^0\right] - (\Delta\mathbf{W}^3 + \Delta\mathbf{W}^2 + \Delta\mathbf{W}^1) \\
\mathbf{W}^{n+1} &= \mathbf{W}^3 + \Delta\mathbf{W}^4
\end{aligned} \tag{18}$$

where $[LHS] = [I] + \alpha\frac{\Delta t}{S}([A]_1\Delta y_1 + [A]_3\Delta y_3 - [B]_1\Delta x_1 - [B]_3\Delta x_3)$ and $\Delta\mathbf{W}^S = \mathbf{W}^S - \mathbf{W}^{S-1}$. Although the *RHS* of stages three and four contain more than one vector $\Delta\mathbf{W}$, only one $\Delta\mathbf{W}$ needs to be stored. Subsequent $\Delta\mathbf{W}$'s are simply added to the stored vector to give $\sum_{i=1}^{s-1}\Delta\mathbf{W}$. The system (18) can be efficiently inverted with a block tridiagonal Gauss elimination routine.

The time step in the semi-implicit integration is limited only by the tangential flux terms, which were treated explicitly. Equation 14 reduces to:

$$\Delta t = \lambda\frac{\Delta l_t}{u_t + C}\ , \tag{19}$$

where l_t is the tangential spacing, and $\lambda \le 2\sqrt{2}$ [Loyd et al. 87]. The normal spacing is no longer restrictive.

4.3 Matrix Conditioning

For grids with high aspect ratio cells $(\Delta x/\Delta y >> 1)$ the matrices [LHS] become increasingly ill conditioned. Consider, for example, a rectangular mesh with $\Delta x = Const.$ and $\Delta y = \Lambda\Delta x$, where $\Lambda << 1$. Then,

$$\begin{aligned}
[\mathrm{LHS}] &= [I] - \alpha\tfrac{\Delta t}{\Lambda\Delta x^2}\left[[B]_1\Delta x_1 + [B]_3\Delta x_3\right] \\
&= [I] + \alpha\tfrac{\Delta t}{\Lambda\Delta x}\left[[B]_{j+1} - [B]_{j-1}\right]\ ,
\end{aligned} \tag{20}$$

and diagonal dominance is lost as the off diagonal terms increase with $1/\Lambda$.

We increase the diagonal dominance of $[LHS]$ by adding implicit smoothing. It is applied by adding to $[LHS]$ the undivided second difference operator:

$$-\,\mu_{IS}\left[\mathbf{W}_{j+1} - 2\mathbf{W}_j + \mathbf{W}_{j-1}\right]\ . \tag{21}$$

The implicit smoothing does not affect the steady state solution; however, large μ_{IS} may inhibit convergence to steady state.

4.4 Residual Smoothing

The Courant number limitation both for the explicit and semi-implicit schemes can be relaxed by smoothing the residuals. In effect, this increases the stencil of influence of the difference scheme and

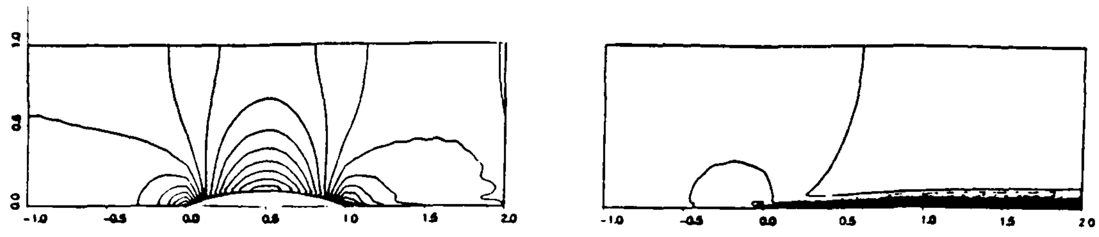


Figure 2: Mach Contours in Channel Flow: Inviscid & Viscous ($Re = 2000$)

thus increases the permissible time step. Convergence acceleration is a result of both the increase in Δt and the damping of the residuals. Residual smoothing is best applied implicitly. In two dimensions:

$$\begin{aligned}(1-\mu_{RS}\delta_{yy})\mathbf{R}' &= \mathbf{R} \\ (1-\mu_{RS}\delta_{xx})\mathbf{R}'' &= \mathbf{R}'\end{aligned} \tag{22}$$

where $\mathbf{R}$ is the vector of residuals and δ is the undivided second difference operator. In the semi-implicit scheme residual smoothing is applied only in the explicit direction, since the implicit discretisation and smoothing has a similar effect in the cross stream direction. Optimal values of the smoothing coefficient and the new time step may be found by numerical experimentation.

4.5 Multi-Grid Convergence Acceleration

Multigrid can also be used to accelerate convergence. Multigrid works by accelerating the propagation of information across the grid. The conservation equations are solved on successively coarser grids to achieve, in effect, a larger difference stencil. Coarse corrections are interpolated back up to the fine grid which drives the scheme. On coarse grids, substantial 2nd and 4th difference damping is added to the fluxes to kill short wave length disturbances. The restriction operator, or forcing function, is also smoothed with a second difference operator. We use a simple V-type strategy to cycle from fine through coarse grids and back to fine. Details are given in [Loyd et al. 87].

5 Results

Following are results of a set of inviscid and viscous flow cases. Figure 2 gives the geometries and typical Mach number contours of the two sets of cases. The inviscid case is a $M_\infty = .5$ channel flow with a $t/c = 0.1$ circular arc bump. The viscous test case is laminar flow at $M_\infty = 0.5$ in a channel, with a lower wall beginning 1/3 of the way into the channel. Symmetry is assumed at the upper boundaries. Because both flows are subsonic no second difference smoothing is added to the spatial operator. A very small amount of fourth difference smoothing ($\nu_4 = 0.002$) is added.

The cases were run with each of the three schemes. Considerable care was taken to make a fair comparison, and for each method, parameters used were those that gave the most efficient solution. All three codes were written from scratch by the first author. The explicit scheme simply requires setting $[B] = [A] = 0$ in Equation 18. The Beam & Warming scheme uses a three point backward temporal integration, although backward Euler temporal integration was also tried with similar results. The *CFL* number resulting in quickest convergence was chosen. Boundary conditions, smoothing formulation, and flux balance formulations in the codes are identical.

Convergence was taken as

$$\frac{1}{I\times J}\sum_{i,j=1}^{I,J}\left\{\left|\frac{\Delta\rho u}{\Delta t}\right| + \left|\frac{\Delta\rho v}{\Delta t}\right| + \left|\frac{\Delta E/E_\infty}{\Delta t}\right|\right\}_{i,j} \le 5\times 10^{-4}\,. \tag{23}$$

This criterion allows fair comparison of convergence histories calculated with different methods and time steps. All calculations were made on a *DEC Microvax II* which is approximately equivalent to a *VAX 750.*

5.1 Inviscid Flow Cases

Table 1 gives the iteration histories for the inviscid cases calculated with the explicit scheme. The second and third columns give the *CPU* time (in minutes) and iteration count for the scheme without acceleration devices. The fourth and fifth give those results for the scheme with multigrid (MG) and residual smoothing (RS). Each calculation was made on a grid with 48 streamwise cells and 16, 24, or 32 cells in the cross stream direction.

Table 1: Inviscid Solution with Explicit Scheme

	NO. ACCEL.		MG & RS	
# cells	CPU(m)	ITER	CPU(m)	ITER
16	291	4177	19	155
24	754	7295	37	208
32	1370	9970	71	306

Solutions without acceleration mechanisms are characterised by a very slow convergence rate, due, in part, to slow damping of pressure waves. Multigrid and residual smoothing are are very effective for this case, resulting in an iteration count reduction of up to a factor of 35 which gives a factor of 20 savings in *CPU* time.

Table 2 gives convergence histories for the same flow cases using the semi-implicit and fully implicit schemes. The iteration count is much smaller with both the semi-implicit and the fully implicit algorithm than with the explicit scheme. However, each iteration takes proportionally more *CPU* time, resulting in similar efficiency. Note that *SINSS* iteration counts, especially, appear to be unaffected by the number of normal grid cells.

Table 2: Inviscid Solutions with *SINSS* and Beam & Warming Scheme

	S-I		B & W	
# cells	CPU(m)	ITER	CPU(m)	ITER
16	42	118	30	190
24	63	117	62	264
32	83	115	63	203

5.2 Viscous Flow Cases

In viscous flow cases, the accurate prediciton of skin friction is usually of importance. To ensure that this quantity is converged we require, *in addition* to (23), that the percentage change in skin friction coefficient over time is small:

$$\sum_{n=i}^{i-2} \left[\frac{\%\Delta(C_f)}{\Delta t_{ave}} \right]^n \leq 5 \cdot 10^{-4} \qquad (24)$$

where $\%(\Delta C_f) = (C_f^i - C_f^{i+1})/C_f^i$ and Δt_{ave} is a representative time step. Summing over three iterations helps eliminate spurious small values of $\Delta\%C_f$ due to oscillatory convergence of the skin friction coefficient.

Reynolds numbers (based on channel height) are 2×10^3, 10^4, and 10^5. All cases have 48 cells in the streamwise direction and 24, 32, or 32 cells across the half-channel. The grids were generated with stretchings $A = \Delta y_{j+1}/\Delta y_j$ of 1.12, 1.15, or 1.18, respectively.

Iteration histories and CPU requirements are given in Table 3. The second column gives the acceleration mechanism (RS and/or MG) for the explicit code. The semi-implicit code used only residual smoothing.

SINSS does significantly better than the explicit or the implicit scheme at all Reynolds numbers. However, at $Re = 10^5$ an implicit smoothing coefficient value of $\mu_{IS} = 0.2$ was necessary for convergence. The implicit scheme converged with difficulty at the highest Reynolds number, despite attempts with a

Table 3: Viscous Channel Flow

	EXPLICIT			S-IMPLICIT		B & W	
Re #	Accel.	CPU(m)	ITER	CPU(m)	ITER	CPU(m)	ITER
2×10^3	MG & RS	99	493	66	109	75	298
	no acc.	219	1678	71	118	-	-
1×10^4	MG & RS	366	1385	69	85	142	421
	no acc.	574	3345	98	121	-	-
1×10^5	RS	548	2820	136	168	550	1500
	no acc.	758	4418	232	287	-	-

variety of parameter values. The grid stretching at high Reynolds numbers renders the explicit solver increasingly stiff and decreases the effectiveness of multigrid. The Reynolds number 10^5 case converged only *without* multigrid.

6 Conclusions

A semi-implicit algorithm for solving the thin layer Navier-Stokes equations in finite volume form is presented. The method retains much of the flexibility of explicit schemes while eliminating the numerical stiffness due to the disparate physical scales found in typical viscous calculations. It is applied to viscous and inviscid flows and is compared to a fully implicit scheme and an explicit scheme equipped with multigrid and residual smoothing. While the explicit scheme is slightly more efficient for inviscid solutions on coarse grids, the semi-implicit algorithm is up to 5 times more efficient than the explicit and implicit schemes on the computed viscous cases.

The algorithm is easy to implement on vector and parallel architecture machines, and preliminary calculations with a fully vectorized code have been made on a *Cray XMP*. An attractive possibility is to use an explicit solver in the outer inviscid flow coupled to *SINSS* in the viscous layer. Future computations will focus on turbulent flows on highly stretched grids and extension to three dimensions.

Acknowledgements

We would like to acknowledge stimulating technical discussions with Bob Melnick and Frank Marconi of Grumman Aerospace Corporation, Bernard Mueller of *FFA*, and Michael Giles of MIT. We gratefully acknowledge *NSF* fellowship support to the first author for the duration of this work, and support by Grumman Aerospace Corporation and *AFOSR* Grant # 82-0136, monitored by Jim Wilson.

References

[Beam & Warming 76] Beam, R.W. and Warming, R.F., "An Implicit Factored Scheme for the Compressible Navier-Stokes Equations", *AIAA* Journal Vol. 16, pp. 393-401.

[Jameson et al. 81] Jameson, A., Schmidt, W., and Turkel, E. "Numerical Solutions of the Euler Equations Using Runge-Kutta Time Stepping Schemes", *AIAA* Paper 81-1259, 1981.

[Loyd et al. 86] Loyd, B., Murman, E. M., and Abarbanel, S. S., "A Semi-Implicit Scheme for the Navier Stokes Equations", Presented at the Society of Industrial and Applied Mathematics National Meeting, Boston, 24 July 1986.

[Loyd et al. 87] Loyd, B., Murman, E. M., and Abarbanel, S. S. "Semi-Implicit Solution of the Navier-Stokes Equations (*in preparation*)", CFDL Report #87-7, M. I. T., 1987.

APPENDIX 4

AIAA'88

AIAA-88-0034

Adaptive Finite Element Methods for the Euler Equations

Richard A. Shapiro
Earll M. Murman
Massachusetts Institute of Technology
Cambridge, MA

AIAA 26th Aerospace Sciences Meeting

January 11-14, 1988/Reno, Nevada

Adaptive Finite Element Methods for the Euler Equations

Richard A. Shapiro[*] and Earll M. Murman[†]

Computational Fluid Dynamics Laboratory

Department of Aeronautics and Astronautics

Massachusetts Institute of Technology

1 Abstract

An adaptive finite element algorithm for solving the steady Euler equations is presented. The algorithm uses quadrilateral elements and allows embedded meshes. An analysis showing how many finite volume and finite difference methods can be viewed as finite element methods is presented, and the methods are compared. A higher-order approximation based on biquadratic interpolation is introduced, and its usefulness is demonstrated. Examples including channel flow and flow in a scramjet inlet demonstrate the utility of unstructured grids, adaptation, and higher-order elements.

2 Introduction

Numerical solution of the Euler equations describing the dynamics of an inviscid, compressible, ideal gas are becoming an important tool for the practicing aerodynamicist [1]. Many algorithms have been proposed for the solution of the Euler equations under various names (cell-centered finite volume, node-centered finite volume, finite element to name but a few), and many authors argue the virtues of each approach. We believe that the important distinction is not finite volume vs. finite element vs. finite difference, but rather structured mesh vs. unstructured mesh. For the remainder of this paper, we will refer to any unstructured mesh algorithm as a finite element algorithm. The main advantage of finite element methods is geometric flexibility. They allow complex geometries to be treated in a straightforward manner, and allow for the use of grid adaptation.

In this paper, we examine some of the differences and similarities of various formulations of the finite element method. Section 3.1 describes three algorithms, which we call Galerkin [2,3,4], cell-vertex, and central difference, and shows how they fit into the finite element framework. For example, on grids with parallellogram elements, the central difference algorithm described is equivalent to Jameson's cell-centered finite volume method [5]. The cell-vertex method [6,7] gives the same difference stencil as a node-based finite volume scheme, or as the first step of a Ni Scheme [8]. We hope this will reduce the confusion in finite element/volume/difference *nomenclature* and put the focus on the *algorithms*.

We also examine the use of higher-order (biquadratic) elements. We have developed a Galerkin formulation using biquadratic interpolation functions, and demonstrate its usefulness for typical problems. A mesh of biquadratic elements requires many fewer elements in smooth regions of a flow, but can have mild problems near discontinuities. We are exploring the use of mixed bilinear and biquadratic elements.

Much work has been done on the use of adaptive grids, either by embedding [4,9,10,11], grid regeneration [12,13], or grid redistribution. We explore further the use of adaptive gridding, introducing and expanding the idea of directional embedding as proposed by Kallinderis and Baron [14].

Next, we show some examples of the ideas presented here. The basic examples we use are the flow over a 4% circular arc bump at Mach 1.4 and the flow over a 10% circular arc bump at Mach 0.68. We show some comparisons between the Galerkin, cell-vertex and central difference methods, and demonstrate the usefulness of the biquadratic element. We also demonstrate adaptive gridding for a scramjet inlet, and we show how adaptation can reduce a problem's sensitivity to a poor initial grid. Finally, we demonstrate directional embedding.

3 Solution Algorithm

In this study, the two-dimensional Euler equations describing the flow of an inviscid, compressible fluid are considered. To allow the capture of shocks and other discontinuous phenomena, the Euler equations are written in con-

[*]Research Assistant, Member AIAA
[†]Professor, Fellow AIAA

servative vector form as

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho e\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ \rho uh\end{bmatrix}+\frac{\partial}{\partial y}\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ \rho vh\end{bmatrix}=0 \quad (1)$$

or

$$\frac{\partial \vec{U}}{\partial t}+\frac{\partial \vec{F}}{\partial x}+\frac{\partial \vec{G}}{\partial y}=0 \quad (2)$$

where e is total energy, p is pressure, ρ is density, u and v are the x and y flow velocities, $\vec{U}$ is a vector of state variables, $\vec{F}$ and $\vec{G}$ are flux vectors in the x and y directions, and h is the total enthalpy, given by the thermodynamic relation

$$h=e+\frac{p}{\rho}. \quad (3)$$

In addition, one requires the equation of state

$$\frac{p}{\rho}=(\gamma-1)\left[e-\frac{1}{2}(u^2+v^2)\right] \quad (4)$$

where the specific heat ratio γ is taken as a constant (1.4) for all calculations reported.

3.1 Spatial discretization

The finite element approach to discretizing these equations divides the domain into elements determined by some number of nodes (in the current implementation, 4 nodes make up a bilinear element, and 9 nodes make up a biquadratic element). These elements can be of any shape. The shapes most commonly used are triangles and quadrilaterals. In this paper, we use only quadrilateral elements. A quadrilateral (or hexahedral) grid will require fewer elements for a given number of nodes than will a triangular (or tetrahedral) grid, and hence will be less expensive to compute and/or require less memory. Also, quadrilateral elements are slightly more accurate [15]. However, there is nothing inherent in the formulation of the algorithm presented here that is limited to a particular element topology.

Within each element the state vector $U^{(e)}$ and flux vectors $F^{(e)}$ and $G^{(e)}$ are written

$$U^{(e)} = \sum N_i^{(e)}U_i^{(e)} \quad (5)$$
$$F^{(e)} = \sum N_i^{(e)}F_i^{(e)} \quad (6)$$
$$G^{(e)} = \sum N_i^{(e)}G_i^{(e)} \quad (7)$$

where the $U_i^{(e)}, F_i^{(e)}, \text{and} G_i^{(e)}$ are the nodal values of the state vector in element e and the $N_i^{(e)}$ are a set of interpolation functions on that element. These interpolation functions have the following properties:

1. each function is 1 at one node and 0 at all other node in the element,
2. each function is 0 outside the element;
3. the sum of all the interpolation functions in the element is 1 everywhere in the element.

Polynomials are usually used for the interpolation functions. These polynomials are expressed in terms of local coordinates (ξ,η), which are related to (x,y) by an isoparametric transformation. Thus, inherent in the formulation that follows are some transformational metrics, which are not shown for clarity.

These expressions can be differentiated to obtain an expression for the derivative in each element in terms of the nodal values (shown here for the state vectors)

$$\frac{\partial U^{(e)}}{\partial x_j}=\sum_{i=1}^{NNE}\frac{\partial N_i^{(e)}}{\partial x_j}U_i^{(e)} \quad (8)$$

where NNE is the number of nodes in that element. The flux vector derivatives are calculated the same way.

The expression for the derivatives is substituted into equation (2) and summed over all elements to obtain

$$\frac{\partial \vec{U}_i}{\partial t}=-\frac{\partial \vec{N}}{\partial x}\vec{F}_i-\frac{\partial \vec{N}}{\partial y}\vec{G}_i \quad (9)$$

where $\vec{N}$ is now a global vector of interpolation functions, determined by summing the interpolation functions for each element.

The next step can be thought of as a projection onto the space spanned by some other functions N', called test functions such that the error in the discretization is orthogonal to the space spanned by the test functions. For more detail on the mathematics involved see [16]. To do this, multiply Eq. (9) by $\vec{N}'$ and integrate over the entire domain. This results in the semi-discrete equation

$$M\frac{\partial \vec{U}_i}{\partial t}=-\int\!\!\int(\vec{N}'^T\frac{\partial \vec{N}}{\partial x}\vec{F}_i+\vec{N}'^T\frac{\partial \vec{N}}{\partial y}\vec{G}_i)\,dV \quad (10)$$

$$M=\int\!\!\int \vec{N}'^T\vec{N}dV \quad (11)$$

or

$$M\frac{\partial \vec{U}_i}{\partial t}=-R_x\vec{F}_i-R_y\vec{G}_i \quad (12)$$

where M is the consistent mass matrix and R_x and R_y are what we call residual matrices. The mass matrix M is sparse, symmetric, and positive definite, but not structured, so it is replaced by a lumped (diagonal) mass matrix M_L in which each diagonal entry is the sum of all the elements in the corresponding row of M. This allows Eq. 12 to be solved explicitly. The lumping does not change the steady-state solution, but does modify the time behavior of the algorithm.

3.1.1 Choice of Test Functions

Various choices for N' are possible, each giving rise to a particular discretisation. If one choses each $N_i'^{(e)}$ to be the corresponding $N_i^{(e)}$, one obtains the Galerkin Finite Element approximation [4]. This approximation has several interesting features. First, it gives the minimum steady-state error, since there is no component of that error in the space of the interpolation functions.

Second, for bilinear functions, on a uniform mesh, for the Euler equations, in steady-state, it is a fourth-order accurate approximation. Finally, on certain mesh topologies, it can be viewed as a finite volume approximation, where the line integral is taken around a set of 4 elements, assuming that the integrand varies parabolically along each edge.

If one chooses each $N_i'^{(e)}$ to be a constant, on bilinear meshes one obtains the "cell-vertex" approximation [7]. This approximation is 100% identical to a node-based finite volume method.

Finally, if one chooses each $N_i'^{(e)}$ to be a Dirac Delta function at node i, one obtains the central difference or collocation approximation. For the bilinear case on a mesh of parallellograms, this is identical to a cell-based finite volume method. If the mesh is not a mesh of parallellograms, this equivalence does not hold. The practical differences between these methods will be discussed in section 4.

3.2 Wall boundary condition

At walls, the portions of the flux vectors representing convection normal to the wall are set to zero to enforce the flow tangency condition. The equation for the fluxes is then

$$\vec{F}_w = \begin{bmatrix} \rho u_m \\ \rho u u_m + p \\ \rho u_m v \\ \rho u_m h \end{bmatrix} \qquad \vec{G}_w = \begin{bmatrix} \rho v_m \\ \rho u v_m \\ \rho v v_m + p \\ \rho v_m h \end{bmatrix} \tag{13}$$

where u_m and v_m are corrected velocities such that the total convective contribution normal to the wall will be 0. These velocities at each node are

$$u_m = u(1 - n_x^2) - v n_x n_y \tag{14}$$
$$v_m = v(1 - n_y^2) - u n_x n_y \tag{15}$$

where n_x and n_y are the components of the unit normal.

There is an option in the code either to enforce flow tangency after each iteration, or to allow this condition to be reached only in steady state. If flow tangency is enforced, convergence rate are improved slightly, and the robustness is improved greatly. For example, without enforcing flow tangency, we were limited to free stream Mach numbers of about 5.5, but with flow tangency enforced explicitly, we have successfully run problems with free stream Mach numbers as high as Mach 8. We enforce flow tangency in most cases.

3.3 Far-field boundary condition

A one-dimensional characteristic treatment is used on the far field boundary. From the inward-directed unit normal vector $\hat{n}$, the unit tangent vector $\hat{t}$ and the normal and tangential velocities u_n and u_t are calculated. The 1-D Riemann invariants (and the corresponding wave speeds) are

$$\text{invariants:} \begin{bmatrix} \dfrac{2a}{\gamma - 1} - u_n \\ \dfrac{2a}{\gamma - 1} + u_n \\ \dfrac{p}{\rho^\gamma} \\ u_t \end{bmatrix} \qquad \text{speeds:} \begin{bmatrix} u_n - a \\ u_n + a \\ u_n \\ u_n \end{bmatrix}. \tag{16}$$

If there is no entropy variation normal to the boundary, these invariants are exact, otherwise they are approximate. At each point on the boundary, the invariants are calculated using the solution state vector $\vec{U}$ and the free stream state vector $\vec{U}_\infty$. Then a decision is made based on the sign of the corresponding wave speed on whether to use the invariant based on the current state or the invariant based on the free stream. If the relevant wave speed is positive (ex., in supersonic inflow, all 4 characteristic speeds are positive) then the free stream value is used.

The invariants are transformed back into a temporary set of primitive variables, and these primitive variables are used to calculate the fluxes for use in the residual calculation. An alternative is to update the state vector based on the new invariants at this time. Updating the state vectors each time the boundary condition is calculated seems to improve the robustness of the method, especially for biquadratic elements.

3.4 Smoothing

To capture shocks and stabilize the scheme, an artificial viscosity needs to be added. Currently, the smoothing used consists of a fourth-difference term and a pressure-switched second-difference term similar to that discussed by Rizzi and Eriksson [17]. Due to the unstructured nature of the grids, a Laplacian-type of second-difference is used, instead of normal and tangential or ξ and η differences.

The heart of the smoothing method is the calculation of an elemental contribution to a second difference. There are two ways we have explored for doing this. The first method, suggested by Ni [8], is relatively fast, conservative and robust (it is dissipative on any element geometry), but gives

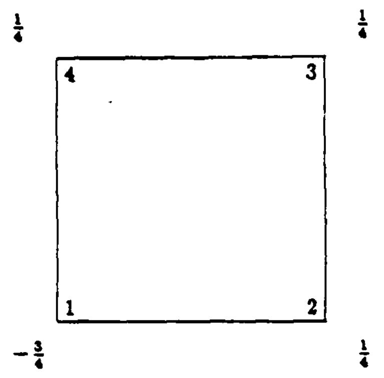


Figure 1: Weights for Second Difference at Node 1

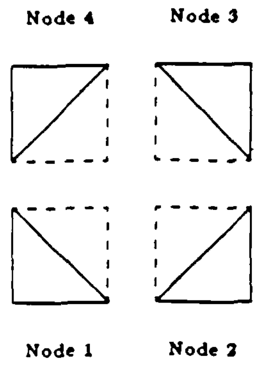


Figure 2: Triangles For Smoothing Calculation

a non-zero contribution to the second difference for a linear function on a non-uniform grid. The second method, proposed by Mavriplis[18] is more expensive, not conservative, less robust (it can be anti-dissipative if the element is not convex), but always results in zero contribution from linear functions on non-uniform grids.

3.4.1 Calculation of a Conservative, Low-Accuracy Second Difference

Figure 1 shows the contribution of a typical element to the second difference at node 1. The numbers inside the box are the node numbers, the numbers outside are the weights. The elemental contribution to a node is obtained by subtracting the value at the node from the average value in the element. The elemental contributions are summed to give the second difference at the node. The elemental contributions can also be multiplied by a scale factor before being summed to the node. This is how the switched second difference is calculated.

3.4.2 Calculation of a Non-Conservative, High-Accuracy Second Difference

This method divides the element into 4 triangles. Figure 2 shows how the element is divided. The figure shows the element in dashed lines, with the triangle outlined in the solid line. The first derivatives are calculated by a line integral around the triangle. Triangle integration is used because the stencil that results from the integration around the entire quadrilateral does not damp the double-sawtooth eigenmode of the residual operator. This derivative is integrated again around an appropriate polygon to get the second difference. The polygon and the integration direction are shown in Fig. 3 for a node in the interior and a node on the boundary. Note that each dashed box represents an element.

At node 1, for example, the contribution from an interior element is

$$
\begin{aligned}
U_{xx} + U_{yy} &= \frac{y_4 - y_2}{2A}\left[U_1(y_2 - y_4) + U_2(y_4 - y_1)\right. \\
&\quad \left. + U_4(y_1 - y_2)\right] \\
&\quad + \frac{x_4 - x_2}{2A}\left[U_1(x_2 - x_4)\right. \\
&\quad \left. + U_2(x_4 - x_1) + U_4(x_1 - x_2)\right]
\end{aligned} \tag{17}
$$

where A is the area of triangle 1-2-4, and (x_i, y_i) are the coordinates of the ith node. For an element on the boundary, the term in front, corresponding to the second integration is changed. For example, for an element with the 1-2 face on a boundary, the term in front would be $x_4 - x_1$ instead of $x_4 - x_2$, corresponding to an integration around two sides of the triangle instead of one. Note that only one factor of A is used, since we want a second difference, not a second derivative.

3.4.3 Combined smoothing

To calculate the complete smoothing for a time step, we first calculate the nodal second difference of pressure by either of the 2 methods. This is turned into an elemental quantity by simple averaging. The elemental second difference is then normalized by an elemental pressure average to form an elemental switch. The second-difference smoothing term is the weighted second difference using the first method above, multiplied by a constant between 0 and 0.05. The fourth-difference smoothing term is the second difference (by the first method) of the second difference (by either method) multiplied by a constant between 0.001 and 0.05. The sum of these two terms is added directly into the time

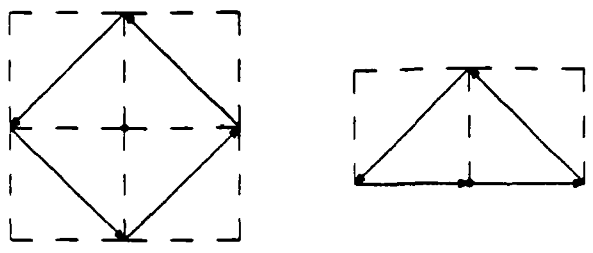


Figure 3: Integration Polygons for Smoothing Calculation

integration of Eq. 18. The smoothing is globally conservative, i.e., the total contribution over the entire domain is zero. This can result in a convective character near boundaries, but this does not affect the solutions adversely. The smoothing of section 3.4.2 is used in all test cases following except for the scramjet calculations, which use the smoothing of Section 3.4.1.

The choice of smoothing method is of great importance. Calculations done by Lindquist and Giles [19] indicate that the accuracy of a complete method for solving the Euler equations can depend more on the smoothing than on the basic difference algorithm. We have done tests and have confirmed this. For instance, the Galerkin and cell-vertex methods are second order accurate when the high accuracy smoothing is used, and both are first order accurate when the low order smoothing is used. This indicates to us that there is a need for further study of artificial viscosity models.

3.5 Time Integration

To integrate equation (12), the following multi-step method is used:

$$\begin{aligned} U_i^{(1)} &= U_i^n + \frac{1}{4}\lambda(\frac{\Delta t_i}{M_{L_i}} R_i(U^n) + V_i^n) \\ U_i^{(2)} &= U_i^n + \frac{1}{3}\lambda(\frac{\Delta t_i}{M_{L_i}} R_i(U^{(1)}) + V_i^n) \\ U_i^{(3)} &= U_i^n + \frac{1}{2}\lambda(\frac{\Delta t_i}{M_{L_i}} R_i(U^{(2)}) + V_i^n) \qquad (18) \\ U_i^{(4)} &= U_i^n + \lambda(\frac{\Delta t_i}{M_{L_i}} R_i(U^{(3)}) + V_i^n) \\ U_i^{n+1} &= U_i^{(4)} \end{aligned}$$

where $R_i(U)$ is the right-hand side of Eq. (12) with the fluxes based on state vector U, V_i is a smoothing term (described above), M_{L_i} is the entry in the lumped mass matrix for node i, and λ is the CFL number. Local time stepping is used to accelerate convergence, with the time step given by

$$\Delta t_i = \frac{\Delta x_i}{|u| + a} \qquad (19)$$

where Δx_i is some nodal characteristic length, and u is the flow velocity at the node. In our algorithm, for Δx_i we use the minimum (over all elements containing the node) of the average lengths of opposite sides of the element.

In one dimension, for stability λ must be less than $2\sqrt{2}$. In two dimensions, a linear stability analysis of the wave equation on a uniform mesh gives the following stability limits: λ must be less than 1.93, for the Galerkin method, 2.17, for the cell-vertex method, and 1.41 for the central difference method. In practice, this estimate is a very conservative one, since the Δx_i we calculate is usually smaller than the characteristic length limiting the stability. In the current implementation, the smoothing is computed at the first stage and is "frozen" for remaining stages.

4 Comparison of the Methods

This section compares the various methods (Galerkin, cell-vertex, and central difference) and discusses some of the important differences between them. An extensive analytic comparison of the methods will be presented in [20], so we will only present the important points here.

4.1 Analytic Comparison

A significant source of error in the solution of flows with supersonic regions is the dispersive error associated with the discrete system of equations. This error appears as low-frequency oscillations either ahead or behind shocks, depending on Mach number and element aspect ratio. For the grids used in the problems, these oscillations should be ahead of the shock for the cell-vertex scheme where the local Mach number is greater than $\sqrt{2}$, and for the central difference and Galerkin methods when the Mach number is less than $\sqrt{2}$. Conversely, the oscillations should be behind the shock for the cell-vertex scheme if the local Mach number is less than $\sqrt{2}$, and for the central difference and Galerkin methods where the Mach number is greater than $\sqrt{2}$. These effects are most apparent when the artificial damping is small. For most practical problems, the damping is large enough to swamp out these errors.

4.2 Numerical Comparison

To see the practical differences between the methods, two test problems were run with each of the 3 methods on the same grid. The test cases were Mach 0.68 flow in a channel over a 10% circular arc bump on a 60x20 grid and Mach 1.4

flow in a channel over a 4% circular arc bump on 60x20 and 90x30 grids.

In the transonic case, there is very little difference in the solutions. Very near the shock, there is a small amount of dispersive error, but this is rapidly killed off because the flow is supersonic only over a small region. Figure 6 shows the Mach number along the bump for the three methods.

In the supersonic case, coarse mesh, the three methods exhibit different behavior on the surface. Figures 7-9 show the surface Mach numbers for the coarse grid case. The solutions are quite different (note the plateau-like feature after the leading-edge shock for the cell-vertex case) and not very good. Interestingly, the difference between the methods is much smaller at mid-channel. Figure 10 shows that in middle of the channel, the solutions are almost identical.

The differences for the supersonic case also are reduced with increasing mesh refinement. Figure 11 shows that for the 90x30 grid, the surface Mach numbers are almost identical again. This illustrates two important points. First, it is important to have enough resolution, and second, if one has enough resolution, there is not a great deal of difference between the methods for these problems.

One important characteristic of each method is the CPU time required to calculate the residual. If one normalizes the CPU time required to calculate everything but the residual (fluxes, smoothing, boundary conditions and updating) to 1 unit, the Galerkin method requires .8 units, the cell-vertex .5 units, and the central difference .9 units for our implementation. Therefore, the total time for a problem can range from 1.5 to 1.9 units. Of course, the CPU timings can vary depending on the exact details of the implementation. We feel that we have implemented the methods in the most efficient manner. It should also be noted that the stability limits for the Galerkin and cell-vertex methods are much better than the limit for the central difference method. When all factors are taken into account, the Galerkin and cell-vertex methods seem to be pretty close to each other, with the central difference method a not-too-close third.

5 Biquadratic Elements

In many problems, the flow may be quite smooth over large portions of the domain. Such problems may benefit from the use of higher-order elements. One element we have been working with is a subparametric *, biquadratic element. Figure 4 shows the element node numbering. The element is subparametric because the positions of nodes 5-9 are functions of the positions of nodes 1-4 (midside nodes are at the midpoint of the side, and the center node is at the "average" position of nodes 1-4). The subparametric restriction allows one to describe the element with 6 geometric parameters, instead of 16 for the full biquadratic element.

*A subparametric element is one in which the geometry is interpolated with a polynomial of lower degree than the state vectors or fluxes

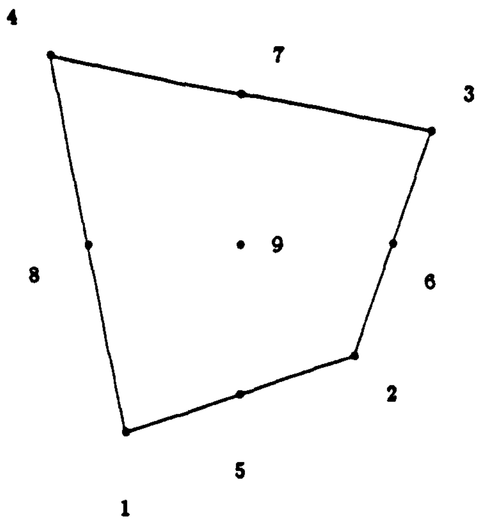


Figure 4: Biquadratic Element Geometry

Element residual calculation is performed exactly as described above for the bilinear elements. We have experimented with several ways to do the smoothing. Most of our efforts have met with failure. Our current method for smoothing calculation is to treat each biquadratic element as four bilinear elements, and to use the method of Section 3.4.1 for calculating the second differences. We plan on implementing the method of Section 3.4.2 but have not yet done so.

The stability limit cannot be computed with the standard Fourier methods, as the difference stencil is different at corner nodes, midside nodes and the center node. Numerical experiments indicate that the CFL limit for the Galerkin method is about 0.8. In the case of the biquadratic elements, the cell-vertex and central difference methods seem to be unstable.

Figures 13 and 15 show the pressure contours for the circular arc bump problems discussed above. The biquadratic calculations were done on a 24x8 grid (Fig. 12). Note that the shocks in both cases are captured well. The surface pressure plots (Figs. 14 and 16) show that the biquadratic elements have larger overshoots at the shocks, as expected. The surface plots also indicate the presence of some spurious high frequency oscillations. The artificial viscosity for the biquadratic elements is still an area for research.

The supersonic case required 272 iterations and 1.2 minutes of CPU time on an Alliant FX/8 computer with 3 CE's. For comparison, a 1200 element bilinear Galerkin case (comparable accuracy) requires about 270 iterations and 2.2 minutes of CPU time. The transonic case converged in 725 iterations and used 3 CPU minutes on the same machine. For comparison, a 1200 element bilinear case took 5 CPU minutes. This program is fully vector-concurrent on the Alliant FX/8, demonstrating the suitability of the algorithm for high performance supercomputers. Due to the unstructed mesh, gather-scatter performance is a major architectural concern.

6 Adaptation

In order to make the best use of computational resources, it is desirable to put the points where the flow has interesting features. This involves detecting the feature, and deciding how to adapt. The details of managing the grid data structure during adaptation are too involved to be discussed here.

6.1 Adaptation Procedure

In a typical problem, one starts out with an idea of what the flow will look like, but one may not know exactly where the features lie. The adaptive approach starts with a fairly coarse initial grid, coarse enough to be cheap to compute, yet fine enough so that most of the essential features can appear. The first step is to compute a solution on the coarse grid. It is usually not necessary to run this solution to convergence, and we usually run it "halfway" to convergence, that is, until the RMS change is the square root of the converged value. Then, an adaptation parameter is calculated, and cells are flagged either for refinement or unrefinement. The flagged cells at the coarsest level are divided, then those at the next coarsest, etc. After all division is done, unrefinement proceeds from the finest level to the coarsest. After all unrefinement has been done, the grid geometry is recalculated, along with some quantities used for vectorisation. The solution is interpolated onto the new grid, and the calculation proceeds. In a typical problem, an adaptation take about as much time as an 2 or 3 iterations, but since adaptation occurs infrequently, the time involved is not significant.

6.2 Adaptation Criteria

There is a good amount of literature about the best choice of adaptation parameter. Dannenhoffer and Baron [9] suggest using the first difference of density as the criterion for adaptation. Löhner [10] suggests using a more complex indicator, which is essentially a second difference of density scaled by a first difference of density. Berger, et. al.[21] use Richardson extrapolation to estimate the errors. We have found that the first difference of density tends to give better grids for transonic flows, while the scaled second difference works better for flows with many shocks at high Mach number. This bears out the experiences of these authors.

Our first-difference indicator (used for transonic flows) is calculated as follows:

1. In each element, calculate the absolute value of the first difference of density (the exact details are not critical, as long as all elements are treated identically.)
2. Compute the mean and standard deviation of this quantity.
3. Normalise this quantity by subtracting the mean and dividing by the standard deviation. If the standard deviation is small, use some arbitrary value instead (currently 0.05 times the mean or .0005, whichever is larger.)
4. Compute the median, and subtract it from the parameter.
5. If the scaled parameter is greater than some threshold value, try to refine that element. If it less than another threshold value, try to collapse the element.

We have found that refine thresholds near .3 and collapse thresholds near -.3 give good results. The shifting by the median is performed after the normalisation because it is much easier to calculate the median of a quantity when has a better idea of the range. The shifting is necessary because the distribution of the adaptation parameter can become very skewed as the calculation progresses.

Our second-difference based parameter is computed as follows:

1. Compute a nodal second difference of density using one of the methods outlined above.
2. Compute a nodal first difference. Again, the details are not important.
3. Compute an average density at the nodes.
4. Compute a nodal adaptation switch:

$$\text{switch} = \frac{|\text{second difference}|}{|\text{first difference}| + \epsilon \times \text{average}} \qquad (20)$$

   where ϵ is a small parameter to "smooth" the switch in smooth portions of the flow. Without it, small oscillations in the flow tend to produce large values of the switch.

5. If the switch is greater than some value (usually around 0.15), divide, if it is less than some value (usually around 0.05) undivide.

6.3 Directional Adaptation

In many cases, the features of interest have strong variations in only one direction. In cases like these, it is wasteful to introduce points in the direction along which little variation occurs. This leads to the concept of directional adaptation. The original motivation in [14] was to use this idea for viscous flows with boundary layers. We have tried to extend these ideas to inviscid flows. The main difference is that in the boundary layers, the rapid variations tend to be along grid lines, while in the inviscid flows, the features (mainly shocks) can lie in any direction. We have developed a criterion for determining which way to divide a cell. The first step is to use some criterion for determining which cells to subdivide. After this is done, determine whether to divide into 4 cells or two cells with the following procedure. Calculate

$$\Delta_H = |\rho_4 - \rho_1| + |\rho_3 - \rho_2| \quad (21)$$

$$\Delta_V = |\rho_2 - \rho_1| + |\rho_3 - \rho_4| \quad (22)$$

$$D_H = \frac{\Delta_H}{\Delta_H + \Delta_V} \quad (23)$$

$$D_V = \frac{\Delta_V}{\Delta_H + \Delta_V} \quad (24)$$

If D_H is greater than some threshold (typically .2-.4), divide the cell horizontally (parallel to the 1-2 face). If D_V is greater then some threshold, divide the cell vertically (parallel to the 2-3 face). If both exceed their thresholds, or neither exceeds its threshold, divide the cell into 4 subcells as for regular embedding.

6.4 Embedded Interface Treatment

An important facet of the algorithm is the treatment of interfaces between coarse and fine regions of the grid. Figure 5 shows a typical interface between a locally fine region and a coarser region. The fluxes at the interface node (node 2) are set to the average of the fluxes at nodes 1 and 3 before residuals are calculated. The state vector at node 2 is set to the average of the state vectors at nodes 1 and 3 after each iteration. No other special treatment is applied at nodes 1,2, or 3. Thus, the overhead involved in having interfaces is extremely small, amounting to less than 2% of the total time per iteration for a typical mesh. This step is also fully vectorized, so it will not be a limitation on a faster machine.

6.5 Examples

To illustrate these ideas, we show how adaptation can be used to improve the solution on an intentionally distorted grid. We then demonstrate directional adaptation. Finally, we show how different problems can produce very different final grids from the same initial grid by showing the adapted results from two scramjet inlet calculations, one at Mach 5, and one at Mach 2.85.

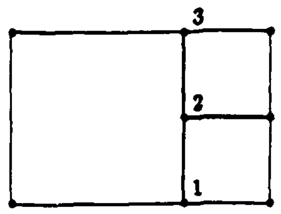


Figure 5: Detail of embedded interface

An advantage of adaptation is that it reduces a problem's sensitivity to the grid. Figure 17 shows a distorted grid for the 10% bump problem. There are places in the grid where the elements nearly degenerate into triangles. After running with 4 adaptations, and a total of 3 different mesh levels, the final grid is shown in Fig. 18. Figures 19 and 20 show the pressure contours and surface pressured on the final grid. These compare well to other solutions on smoother meshes.

To demonstrate directional adaptation, we used as the test example the transonic circular arc bump. The first-difference adaptation criterion was used, the vertical threshold was set to 0.6, and the horizontal threshold was set to 0.8. The final grid is shown in Fig. 21, and contours of pressure in Figure 22. We found that all of the directional embedding results were disappointing. The savings introduced by directionally adapting are small, and the degradation of solution quality is large. The key issue seems to be that in order for directional adaptation to work well, the grids and features must be aligned. This is the case in boundary layers, and has been applied there successfully by Kallinderis and Baron[14].

To show the efficiency gain from adaptation, we computed flow through a scramjet inlet (Fig. 23) from a paper by Kumar [22]. This grid took under 2 hours of human time to generate by the first author, illustrating the utility of finite element methods. In addition, no programming changes were required for these cases. Exactly the same program was used to run the channel flows and the scramjets. If we desired to run a three strut inlet, for example, we would only need to generate the appropriate initial grid. The initial grid for the scramjet is shown in Fig. 23. We ran 2 cases, one at Mach 5.0 and another at Mach 2.85. The Mach 2.85 case was chosen because the flow in the center passage is just about to choke. If the Mach number is reduced further, the inner flow chokes and the inlet can unstart under certain outflow conditions. These cases were run using the cell-vertex method.

Figure 24 shows the final grid for the Mach 5.0 test case. The second-difference based adaptation parameter was used, and the grid was allowed to adapt 5 times. We limited the grid to 3 levels of embedding, mainly because of limitations in the printer that produced the results. At the scale in this paper, anything finer would appear as solid black. We have run one test case where we allowed 4 levels of embedding. Figure 25 shows the density contours in the inlet. Note the clarity of the shocks and expansion fans, and note the bending of the shocks as they pass through the expansions. This is an illustration of a problem that adapts itself almost everywhere. A globally fine grid for this problem would have about 5500 elements, as opposed to 4000 for the adapted grid. The main advantage of the adaptation in this problem is that the solution after each refinement starts out with a guess that is closer to the final answer, so the solution requires fewer iterations to converge.

The next case was a nearly choked inlet. Figure 26 shows the final grid for this case. The Mach number contours are shown in Fig. 27. Notice the pattern of shocks just in front of the throat. There is a very small normal shock right at mid-channel. This is more apparent in the throat close-up, where a distinct subsonic region exists. The minimum Mach number is about 0.96 in that region. As the inflow Mach number is decreased, the subsonic region increases in size until the flow chokes and the solution fails. Fig. 28. Also note the difference compared with the Mach 5 case in shock pattern at the trailing edge and outflow boundaries. Each of these cases required about 12 minutes of CPU time on the Alliant.

Finally, we ran the Mach 5.0 inlet on a grid of the same resolution as the initial grid for the cases just shown (Fig. 23), but with biquadratic elements. Figure 29 shows the density contours for this case. The flow features are only slightly less resolved than the bilinear case (Fig. 25), but this case took only 1.5 minutes to run on the Alliant, a factor of 9 improvement. We feel this illustrates the potential of the biquadratic element, even for flows with shocks.

7 Conclusions

We feel that both the Galerkin and cell-vertex schemes are good, robust methods for solving the Euler equations. For most practical problems there is very little difference between them. We do not recommend the central difference scheme, for stability, efficiency and dispersive reasons. For three-dimensional problems, the cell-vertex method has a much lower operation count.

The techniques and algorithms used here should carry over directly to three dimensions. Work is in progress on a 3-D, hexahedral, adaptive code. A major problem in 3-D is displaying the results.

The directional adaptation has proven to be less effective for inviscid shocked flows than for viscous flows [14]. One possible improvement might be to combine grid motion with directional adaptation. This would reduce a common problem with grid moving schemes; if certain areas of the grid became too sparse, the embedding could fill them in.

It is possible to mix a mesh of bilinear and biquadratic elements in a single problem. This would have the advantage of allowing biquadratic elements to be put in smoother portions of the flow, and bilinear elements near shocks, getting the best of both worlds. The smoothing on biquadratic elements is also under investigation. We are working on an implementation in which the biquadratic element is divided into four sub-elements, each of which is smoothed by the high accuracy method mentioned above.

8 Acknowledgements

This work was supported by the Air Force Office of Scientific Research under contract number AFOSR-87-0218, Dr. James Wilson, technical monitor. The first author wishes to acknowledge the support of the Fannie and John Hertz foundation.

References

[1] A. Jameson, "Successes and Challenges in Computational Aerodynamics," AIAA Paper 87-1184, 1987.

[2] R. Löhner, K. Morgan, J. Peraire, and O. C. Zienkiewicz, "Finite Element Methods for High Speed Flows," AIAA Paper 85-1531, 1985.

[3] K. Bey, E. Thornton, P. Dechaumphai, and R. Ramakrishnan, "A New Finite Element Approach for Prediction of Aerothermal Loads–Progress in Inviscid Flow Computations," AIAA Paper 85-1533, 1985.

[4] R. Shapiro and E. Murman, "Cartesian Grid Finite Element Solutions to the Euler Equations," AIAA Paper 87-0559, January 1987.

[5] A. Jameson, W. Schmidt, and E. Turkel, "Numerical Solutions of the Euler Equations by a Finite Volume Method Using Runge-Kutta Time Stepping Schemes," AIAA Paper 81-1259, June 1981.

[6] M. G. Hall, "Cell Vertex Schemes for the Solution of the Euler Equations," Technical Memo Aero 2029, March 1985.

[7] K. Powell, *Vortical Solutions of the Conical Euler Equations*, Technical Report CFDL-TR-87-8, MIT Computational Fluid Dynamics Laboratory, July 1987.

[8] R. H. Ni, "A Multiple-Grid Scheme for Solving the Euler Equations," *AIAA Journal*, Vol. 20, No. 11, November 1982, pp. 1565–1571.

[9] J. F. Dannenhoffer III and J. R. Baron, "Grid Adaptation for the 2-D Euler Equations," AIAA Paper 85-0484, January 1985.

[10] R. Löhner, "The Efficient Simulation of Strongly Unsteady Flows by the Finite Element Method," AIAA Paper 87-0555, January 1987.

[11] J.T. Oden, T. Strouboulis, P. Devloo, L. W. Spradley, and J. Price, "An Adaptive Finite Element Strategy for Complex Flow Problems," AIAA Paper 87-0557, January 1987.

[12] J. Peiro J. Peraire, K. Morgan and O. C. Zienkiewicz, "An Adaptive Finite Element Method for High Speed Flows," AIAA Paper 87-0558, January 1987.

[13] K. Morgan, R. Löhner, J. R. Jones, J. Peraire, and M. Vahdati, *Finite-Element FCT for the Euler and Navier-Stokes Equations*, Technical Report C/R/540/86, Institute for Numerical Methods in Engineering, February 1986.

[14] J. Kallinderis and J. R. Baron, "Adaptation Methods for a New Navier-Stokes Algorithm," AIAA Paper 87-1167, June 1987.

[15] D. Lindquist and M. B. Giles, personal communication.

[16] G. Strang, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, Wellesley, Massachusetts, 1986.

[17] A. Rizzi and L. E. Eriksson, "Computation of Flow around Wings Based on the Euler Equations," *Journal of Fluid Mechanics*, Vol. 148, November 1984.

[18] D. Mavriplis, *Solution of the Two-Dimensional Euler Equations on Unstructred Triangular Meshes*, PhD thesis, Princeton University, June 1987.

[19] D. Lindquist and M. Giles, "A Comparison of Numerical Schemes on Triangular and Quadrilateral Meshes," June 1988, Submitted to 11th International Conference on Numerical Methods in Fluid Dynamics.

[20] R. Shapiro, "Prediction of Dispersive Errors in Numerical Solution of the Euler Equations," March 1988, To be presented at the Second International Conference on Hyperbolic Problems, Aachen, W. Germany.

[21] M. J. Berger and J. Oliger, "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations," *Journal of Computational Physics*, Vol. 53, 1984, pp. 484–512.

[22] A. Kumar, "Numerical Analysis of the Scramjet-Inlet Flow Field by Using Two-Dimensional Navier-Stokes Equations," NASA Technical Paper 1940, December 1981.

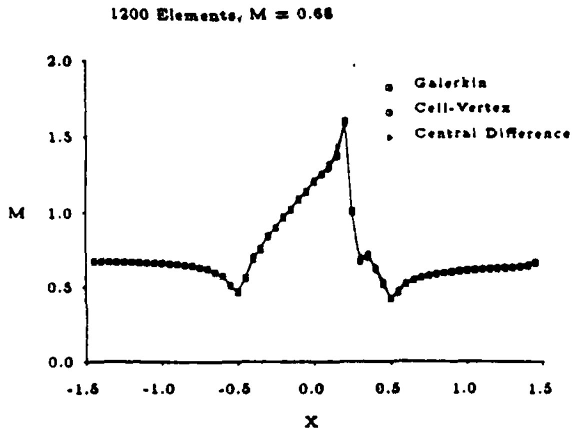


Figure 6: Surface Mach Number Comparisons for the Three Methods, Transonic Case

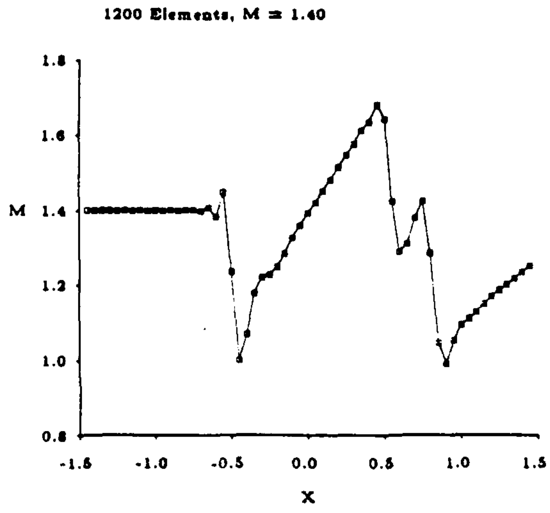


Figure 7: Coarse Grid Surface Mach Number, Galerkin, Supersonic Case

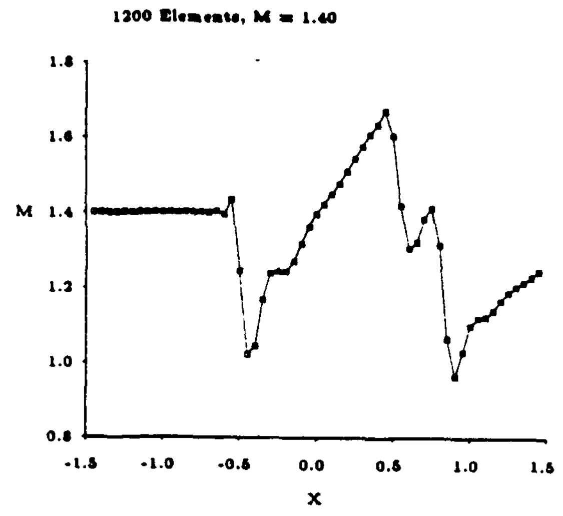


Figure 8: Coarse Grid Surface Mach Number, Cell-Vertex, Supersonic Case

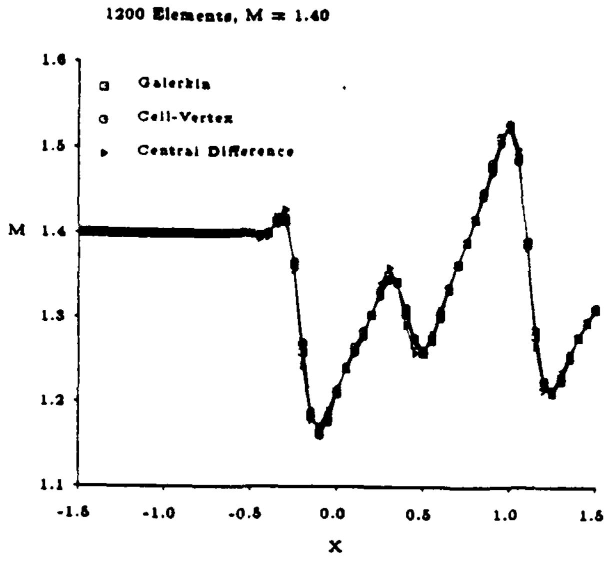


Figure 10: Coarse Grid, Mach Number at Mid-channel, All Three Methods, Supersonic Case

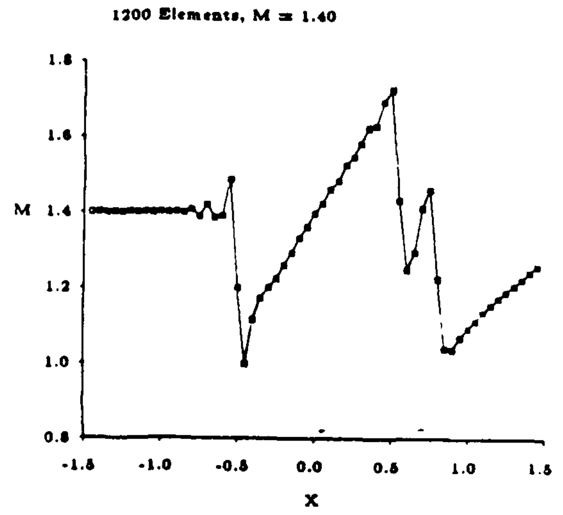


Figure 9: Coarse Grid Surface Mach Number, Central Difference, Supersonic Case

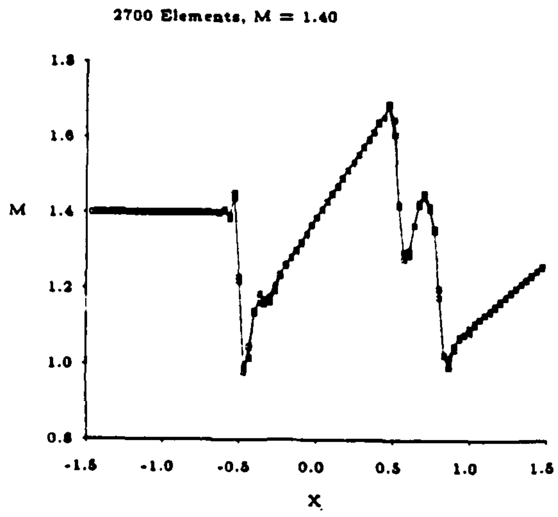


Figure 11: Surface Mach Number Comparisons, Fine Grid, All Three Methods, Supersonic Case

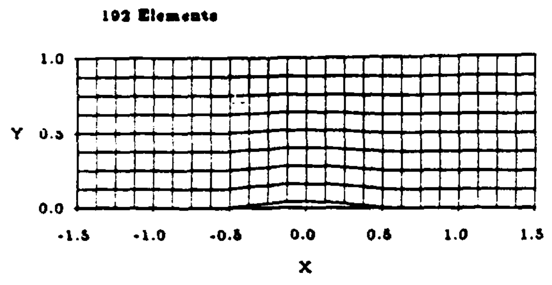


Figure 12: Grid, 4% Circular Arc Bump, Biquadratic Elements–Supersonic Case

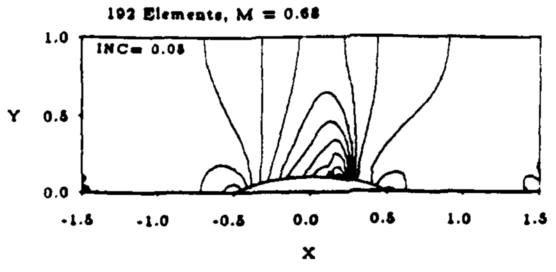


Figure 15: Pressure Contours, 10% Circular Arc Bump, Biquadratic Elements–Transonic Case

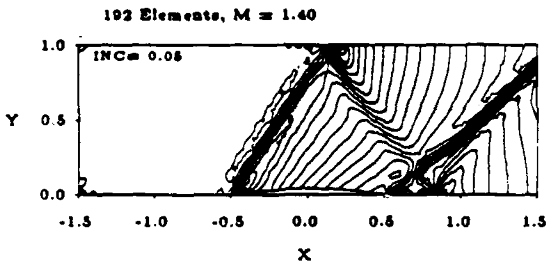


Figure 13: Pressure Contours, 4% Circular Arc Bump, Biquadratic Elements–Supersonic Case

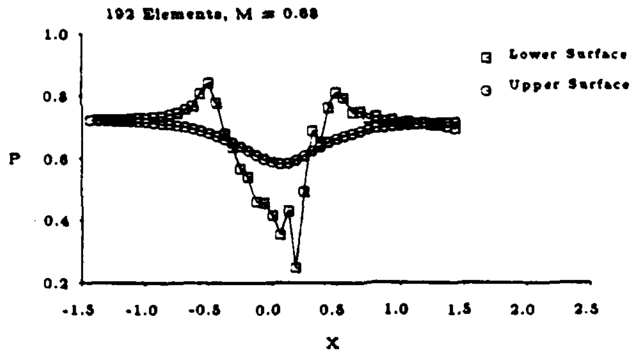


Figure 16: Surface Pressures, 10% Circular Arc Bump, Biquadratic Elements–Transonic Case

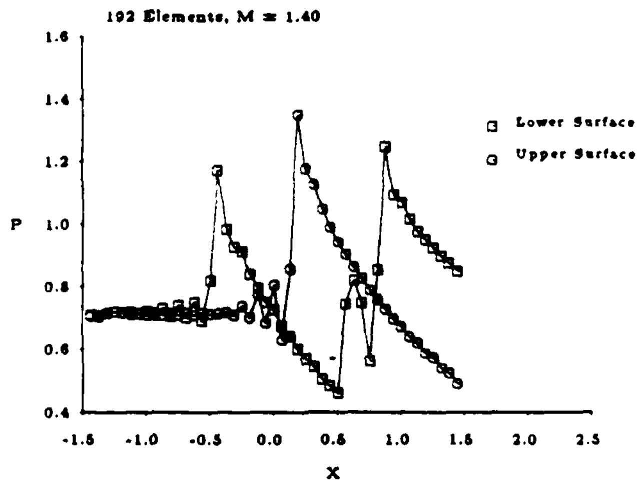


Figure 14: Surface Pressures, 4% Circular Arc Bump, Biquadratic Elements– Supersonic Case

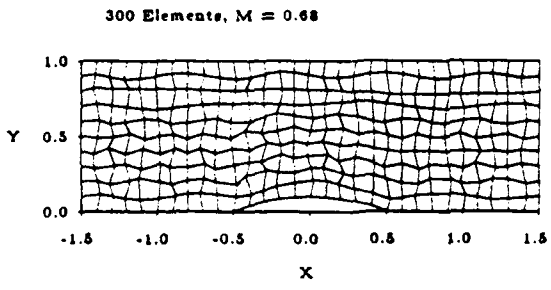


Figure 17: Initial Grid, 10% Circular Arc Bump–Transonic Case

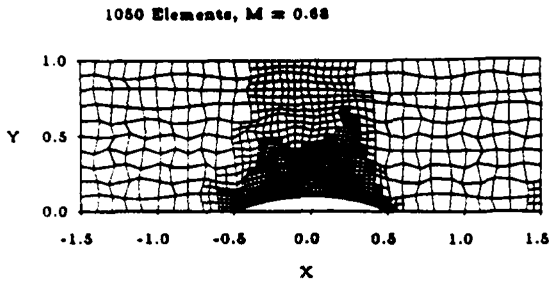


Figure 18: Final Grid, 10% Circular Arc Bump–Transonic Case

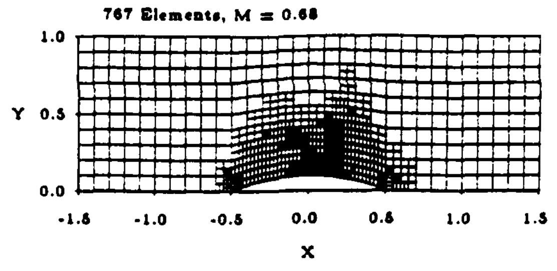


Figure 21: Final Grid, 10% Circular Arc Bump, Directional Adaptation–Transonic Case

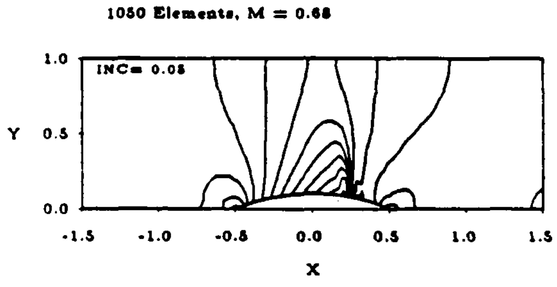


Figure 19: Pressure Contours, 10% Circular Arc Bump, Distorted Grid–Transonic Case

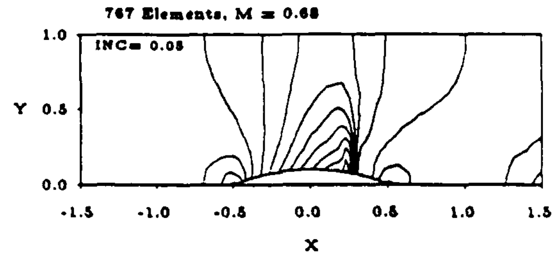


Figure 22: Pressure Contours, 10% Circular Arc Bump, Directional Adaptation–Transonic Case

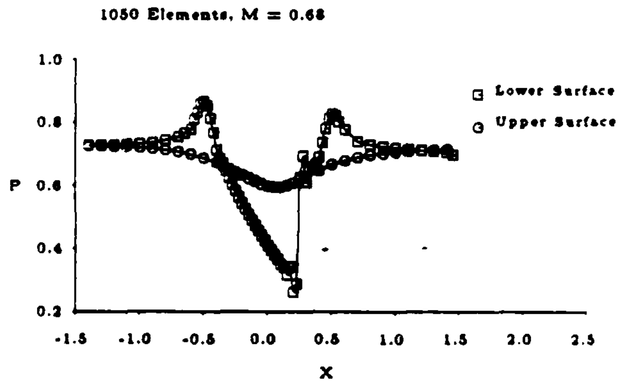


Figure 20: Surface Pressure, 10% Circular Arc Bump, Distorted Grid–Transonic Case

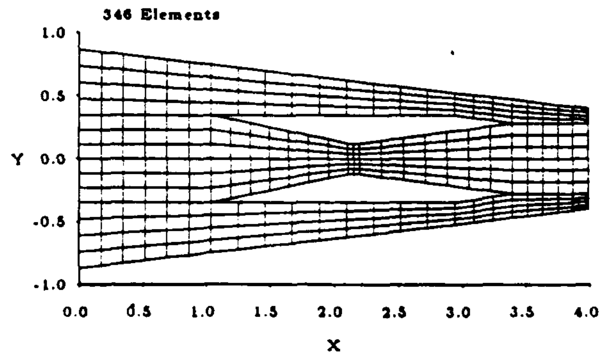


Figure 23: Scramjet Inlet Initial Grid

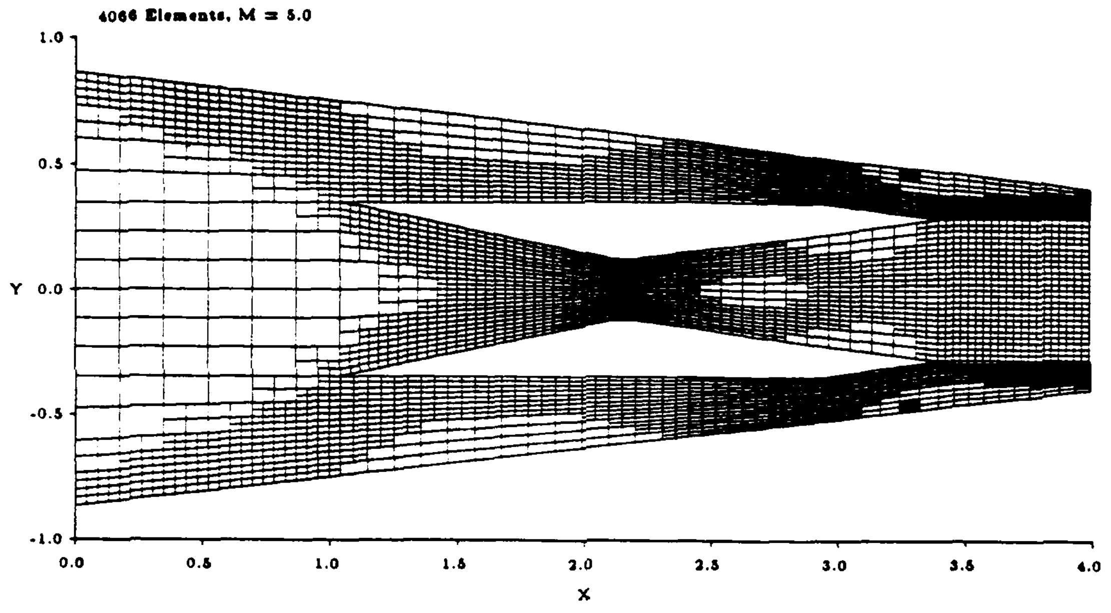


Figure 24: Scramjet Final Grid, M=5.0

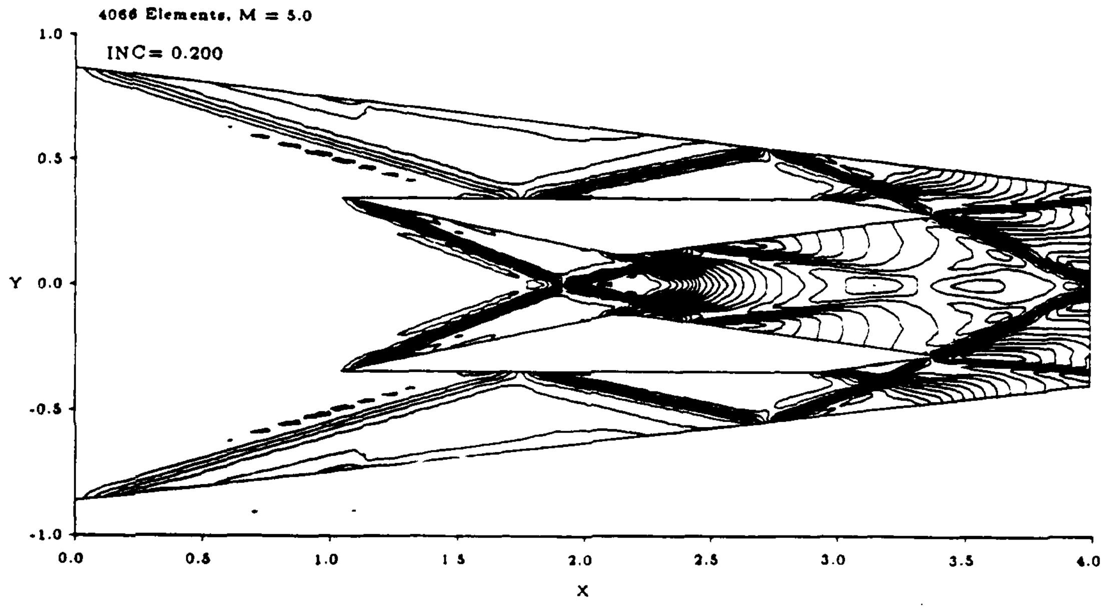


Figure 25: Density Contours, M=5.0

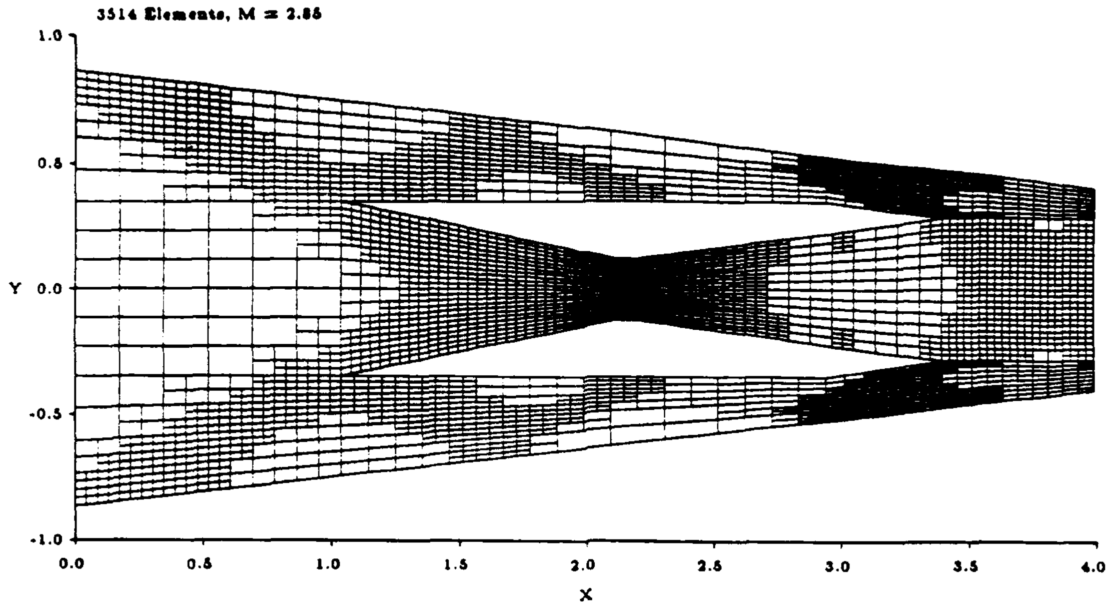


Figure 26: Scramjet Final Grid, M=2.85

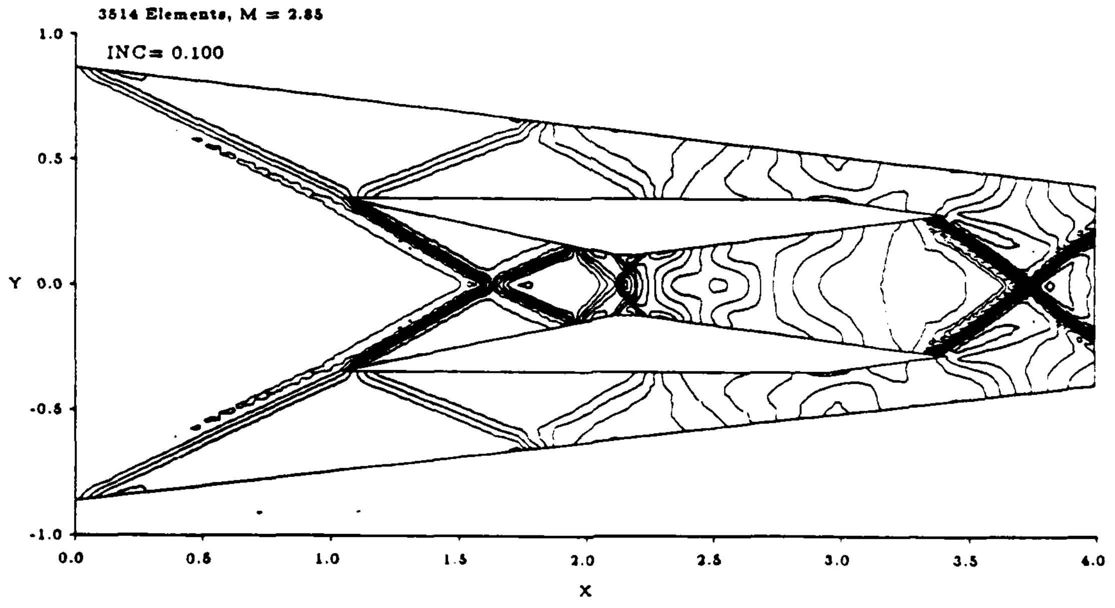


Figure 27: Mach Number Contours, M=2.85

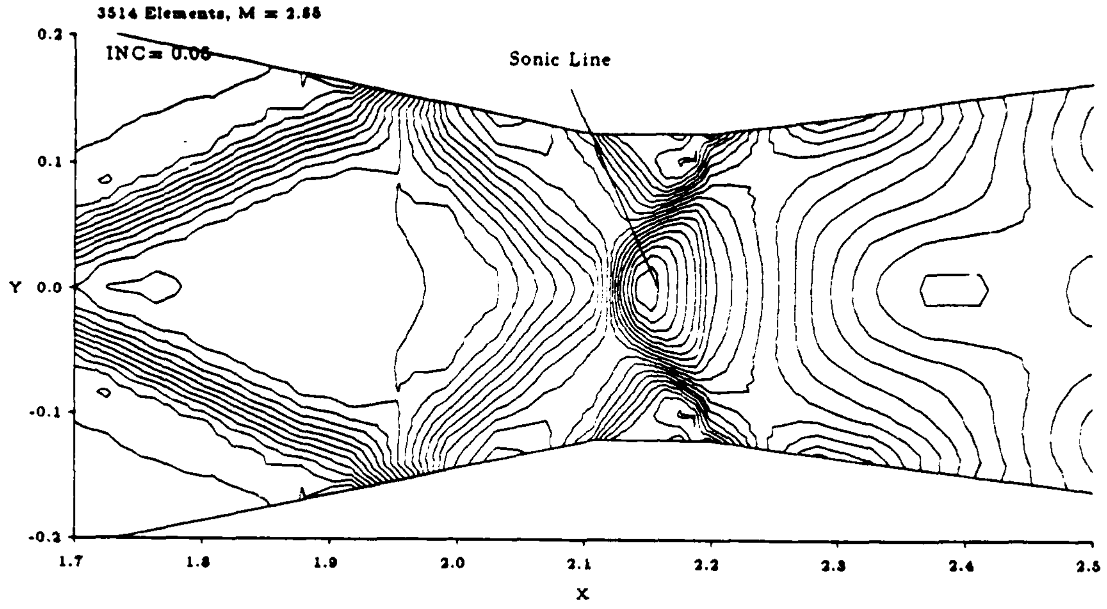


Figure 28: Mach Number in Throat Area, M=2.85

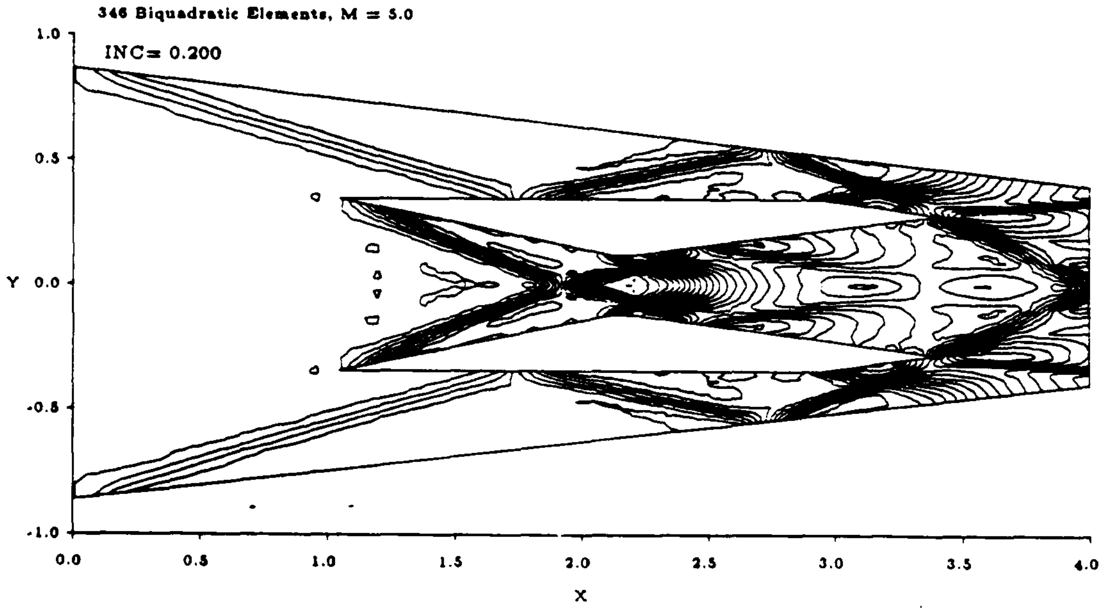


Figure 29: Density Contours, M=5.0, Biquadratic Elements

APPENDIX 5

NASA Contractor Report 181625

ICASE REPORT NO. 88-13

ICASE

COMPACT HIGH ORDER SCHEMES FOR THE EULER EQUATIONS

Saul Abarbanel

Ajay Kumar

Contract No. NAS1-18107
February 1988

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the Universities Space Research Association

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

COMPACT HIGH ORDER SCHEMES FOR THE EULER EQUATIONS

Saul Abarbanel
Tel-Aviv University

and

Ajay Kumar
NASA Langley Research Center

ABSTRACT

An implicit approximate factorization (AF) algorithm is constructed which has the following characteristics.

- In 2-D: The scheme is unconditionally stable, has a 3×3 stencil and at steady state has a fourth order spatial accuracy. The temporal evolution is time accurate either to 1st or 2nd order through choice of parameter.
- In 3-D: The scheme has almost the same properties as in 2-D except that it is now only conditionally stable, with the stability condition (the CFL number) being dependent on the "cell aspect ratios," $\Delta y/\Delta x$ and $\Delta z/\Delta x$. The stencil is still compact and fourth order accuracy at steady state is maintained.

Numerical experiments on a 2-D shock-reflection problem show the expected improvement over lower order schemes, not only in accuracy (measured by the L_2 error) but also in the dispersion.

It is also shown how the same technique is immediately extendable to Runge-Kutta type schemes resulting in improved stability in addition to the enhanced accuracy. [1]

[1]This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-18107 while the first author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23665. Partial support was also provided under USAF Grant No. AFOSR-87-0218.

1 INTRODUCTION

It can be shown [1] that numerical approximations to the linearized Euler equations of gas dynamics give rise to dispersive errors which in the 2-D supersonic case depend on a similarity parameter $\kappa = (\Delta y/\Delta x)\sqrt{M_\infty^2 - 1}$ (under the assumption $v \ll u$ everywhere, where u and v are the x and y components of the velocity vector). The difference between the dispersion relations of any numerical algorithm and that of the original partial differential equations can be plotted as curves in the Fourier plane with κ as a parameter.

In particular the results in [1] indicate that for central-difference schemes, the dispersive error are contributed mostly by the third power of the errors in the Fourier variables θ and ϕ. It is, therefore, natural to think of fourth order spatially accurate algorithms as having better dispersive properties. By utilizing the structure of the Euler equations one can obtain, on a cartesian grid, a fourth order approximation which instead of using a 5×5 stencil (and $5 \times 5 \times 5$ in 3-D) relies on a compact support of 3×3 (and $3 \times 3 \times 3$ in 3-D). The advantages of the combination of fourth order accuracy together with compact support are quite obvious in terms of total computer work and memory.

In Section 2, we describe the construction of an approximate factorization (AF) central difference scheme and examine its theoretical (linear) stability properties. In Section 3, we derive the corresponding 4-step Runge-Kutta scheme and show how the Jameson-Schmidt-Turkel algorithms [2] may be easily modified to that form which has, in addition to the higher accuracy, markedly enhanced stability limits. In Section 4, we describe some numerical experiments using the AF-version. Section 5 summarizes our findings.

2 DERIVATION OF THE APPROXIMATE-FACTORIZATION SCHEME

2.1 The Two-Dimensional Case

Consider a general hyperbolic conservation law in 2-D:

$$\vec{u}_t + \vec{f}_x + \vec{g}_y = 0. \tag{1}$$

In the case of Euler equations, for example, the vectors $\vec{u}, \vec{f}\,(\vec{u}), \vec{g}\,(\vec{u})$ are given by

$$\vec{u} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E \end{pmatrix} \quad \vec{f} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ u(E+p) \end{pmatrix} \quad \vec{g} = \begin{pmatrix} \rho v \\ \rho v u \\ \rho v^2 + p \\ v(E+p) \end{pmatrix} \tag{2}$$

where ρ, u, v, E, and p are respectively the density, velocity components in the x and y directions, the total energy per unit volume and the pressure. In addition there is the equation of state relating algebraically (in the case of gas-dynamics) the internal energy to the pressure and density. One may also write (1) in non-conservation form as

$$\vec{u}_t + A\,\vec{u}_x + B\,\vec{u}_y = 0 \tag{3}$$

where A and B are the Jacobians of $\vec{f}$ and $\vec{g}$ with respect to $\vec{u}$.

Consider first a forward-time second order central finite difference approximation to (1)

$$\frac{u_{j,k}^{n+1}-u_{j,k}^{n}}{\Delta t}+\frac{f_{j+1,k}^{n}-f_{j-1,k}^{n}}{\Delta x}+\frac{g_{j,k+1}^{n}-g_{j,k-1}^{n}}{\Delta y}=0 \tag{4}$$

where we have dropped the sup-arrow indicating a vector and used the conventional differencing notation

$$\vec{u_{j,k}}=u_{j,k}^{n}=u(j\Delta x,k\Delta y,n\Delta t). \tag{5}$$

We have a cartesian grid with nodes at $x=j\Delta x, y=k\Delta y$ and $t=n\Delta t$. We shall now introduce the usual shift operations notation:

$$\begin{aligned}
&D_t u = u_{j,k}^{n+1}-u_{j,k}^{n}\\
&\mu_x u=\tfrac{1}{2}(u_{j+\frac{1}{2},k}+u_{j-\frac{1}{2},k}),\quad \mu_y=\tfrac{1}{2}(u_{j,k+\frac{1}{2}}+u_{j,k-\frac{1}{2}})\\
&\delta_x u=u_{j+\frac{1}{2},k}-u_{j-\frac{1}{2},k},\qquad \delta_y=u_{j,k+\frac{1}{2}}-u_{j,k-\frac{1}{2}}.
\end{aligned} \tag{6}$$

Eq. (4) then may be written as:

$$D_t u=-\lambda\mu_x\delta_x f-\frac{\delta}{I\!R}\mu_y\delta_y g \tag{7}$$

where $\lambda=\Delta t/\Delta x$ and $I\!R=\Delta y/\Delta x$. If we take a Taylor series expansion about the grid point $(x,y,t)=(j\Delta x,k\Delta y,n\Delta t)$, we can construct the modified equation corresponding to (7):

$$u_t+\frac{\Delta t}{2}u_{tt}+\cdots=-\left[f_x+\frac{\Delta x^2}{3!}f_{xxx}+\cdots\right]-\left[g_y+\frac{\Delta y^2}{3!}g_{yyy}+\cdots\right]$$

or

$$u_t+f_x+g_y=-\left[\frac{\Delta t}{2}u_{tt}+\frac{\Delta x^2}{3!}f_{xxx}+\frac{\Delta y^2}{3!}g_{yyy}\right]. \tag{8}$$

Thus if we want to approximate (1) to a higher than second order (and in particular to fourth order in space–see comment in Introduction concerning dispersive errors) we must modify (7) by subtracting out the terms on the right hand side of (8). At this point we realize that using a straightforward approach to approximating f_{xxx} and g_{yyy} will lead to bigger stencils. However, using the original partial differential equation, (1), we have

$$f_{xxx}=-u_{txx}-g_{yxx} \tag{9}$$

and similarly

$$g_{yyy}=-u_{tyy}-f_{xyy}. \tag{10}$$

Since f_{xxx} and g_{yyy} need be approximated only to second order (because of their coefficients, $\Delta x^2/3$ and $\Delta y^2/3$) the required stencil for the terms on the right hand side is only 3×3. (This observation was previously made, in another context by Jones, et al. [3].) So, replacing f_{xxx} and g_{yyy} in (8) by $-\delta_x^2 D_t u/\Delta x^2\Delta t-\delta_x^2\mu_y\delta_y g/\Delta x^2\Delta y$ and $-\delta_y^2 D_t u/\Delta y^2\Delta t-\delta y^2\mu_x\delta_x f/\Delta y^2\Delta x$, respectively; and also replacing u_{tt} by $D_t u_t/\Delta t\rightarrow D_t(-f_x-g_y)/\Delta t\rightarrow -D_t\left[(\mu_x\delta_x f/\Delta x)+(\mu_y\delta_y g/\Delta y)\right]$ we obtain the following approximation to (1) which is spatially fourth order accurate and temporally second order accurate:

$$D_t\left[u+\frac{\Delta t}{2}\left(\frac{\mu_x\delta_x}{\Delta x}f+\frac{\mu y\delta y}{\Delta y}g\right)+\frac{\Delta x^2}{3!}\frac{\delta x^2}{\Delta x^2}f+\frac{\Delta y^2}{3!}\frac{\delta y^2}{\Delta y^2}g\right]$$

$$= -\lambda\mu_x\delta_x f - \frac{\lambda}{3!}\Delta x^3\frac{\mu_y\delta_y\delta_x^2}{\Delta y\Delta x^2}g - \frac{\lambda}{I\!R}\mu_y\delta_y g - \frac{\lambda}{3!I\!R}\Delta y^3\frac{\mu_x\delta_x\delta_y^2}{\Delta x\Delta y^2}f.$$

Using (3) and the definition of D_t we rearrange the above into the delta form

$$\left[I + \frac{\lambda}{2}\left(A\mu_x\delta_x + \frac{B}{I\!R}\mu_y\delta_y\right) + \frac{1}{3!}\left(A\delta_x^2 + B\delta_y^2\right)\right]\left(u_{j,k}^{n+1} - u_{j,k}^n\right)$$

$$= -\lambda\mu_x\delta_x\left(I + \frac{1}{3!}\delta_y^2\right)f_{j,k}^n - \frac{\lambda\mu_y\delta_y}{I\!R}\left(I + \frac{1}{3!}\delta_x^2\right)g_{j,k}^n \tag{11}$$

where I is the identity matrix. This is an implicit unfactored scheme. Since we are interested in marching to steady state we approximate-factor the left hand side to obtain:

$$\left(I + \frac{1}{3!}A\delta_x^2 + \frac{\sigma}{2}\lambda A\mu_x\delta_x\right)\left(I + \frac{1}{3!}B\delta_y^2 + \frac{\sigma}{2}\lambda\frac{B}{I\!R}\mu_y\delta y\right)\left(u_{j,k}^{n+1} - u_{j,k}^n\right)$$

$$= -\lambda\left\{\mu_x\delta_x\left(I + \frac{1}{3!}\delta_y^2\right)f_{j,k}^n + \frac{1}{I\!R}\mu_y\delta_y\left(I + \frac{1}{3!}\delta_x^2\right)g_{j,k}^n\right\}. \tag{12}$$

We introduced into (12) the parameter σ. If $\sigma = 1$ then we retain the temporal second order accuracy while $\sigma = 2$ gives first order in time. Note that (12) involves the inversion of block-tridiagonal matrices. The right hand side represents the steady state operator to fourth-order. The whole scheme involves only a 3 stencil.

2.2 The 3-D Case

The starting point, corresponding to (1) is

$$\vec{u}_t + \vec{f}_x + \vec{g}_y + \vec{h}_z = 0. \tag{13}$$

Following the same steps as in the 2-D case, we get the modified equation

$$u_t + f_x + g_y + h_z = -\left[\frac{\Delta t}{2}u_{tt} + \frac{\Delta x^2}{3!}f_{xxx} + \frac{\Delta y^2}{3!}g_{yyy} + \frac{\Delta z^2}{3!}h_{zzz}\right] \tag{14}$$

where using (13), we have

$$f_{xxx} = -u_{txx} - g_{yxx} - h_{zxx} \tag{15}$$

$$g_{yyy} = -u_{tyy} - h_{zyy} - f_{xyy} \tag{16}$$

$$h_{zzz} = -u_{tzz} - f_{xzz} - g_{yzz}. \tag{17}$$

Repeating all the steps leading to (12) we obtain its three dimensional analog:

$$\left(I + \frac{1}{3!}A\delta_x^2 + \frac{\sigma}{2}\lambda A\mu_x\delta_x\right)\left(I + \frac{1}{3!}B\delta_y^2 + \frac{\sigma}{2}\lambda\frac{B}{I\!R}\mu_y\delta_y\right)\left(I + \frac{1}{3!}C\delta_z^2 + \frac{\sigma}{2}\lambda\frac{C}{Q}\mu_z\delta_z\right)\left(u_{j,k,\ell}^{n+1} - u_{j,k,\ell}^n\right) =$$

$$= -\lambda\left[\mu_x\delta_x\left(I + \frac{1}{3!}\delta_y^2 + \frac{1}{3!}\delta_z^2\right)f_{j,k,\ell}^n + \mu_y\delta_y\left(I + \frac{1}{3!}\delta_x^2 + \frac{1}{3!}\delta_z^2\right)g_{j,k,\ell}^n + \mu_z\delta_z\left(I + \frac{1}{3!}\delta_x^2 + \frac{1}{3!}\delta_y^2\right)h_{j,k,l}^n\right]. \tag{18}$$

In (18) the matrix C is the Jacobian $\partial\vec{h}/\partial\vec{u}$ and the shift operators μ_z, δ_z are defined in a manner analogous to (6). Q is the cell aspect ratio in the z direction, $Q = \Delta z/\Delta x$.

2.3 Stability Analysis for the 2-D Case

We consider the *linearized* (i.e., frozen coefficients) version of (12). We carry out a Von-Neumann analysis in the usual manner by casting (12) in Fourier space. A typical Fourier component of $u^n_{j,k}$ is given by

$$u^n_{j,k} \rightarrow \hat{u}^n e^{ij\phi} e^{ik\theta} \tag{19}$$

where

$$\phi = \ell\Delta x, \quad \theta = m\Delta y \tag{20}$$

with

$$-\pi < \phi, \theta < \pi \quad and \quad -\infty < \ell, m < \infty. \tag{21}$$

The mapping of the various shift operations is as follows:

$$\delta_x u^n_{j,k} \rightarrow 2isin\frac{\phi}{2} \triangleq 2i\xi, \qquad \delta_y u^n_{j,k} \rightarrow 2isin\tfrac{\theta}{2} \triangleq 2i\eta \tag{22}$$

$$\mu_x u^n_{j,k} \rightarrow cos\frac{\phi}{2} = \sqrt{1-\xi^2}, \quad \mu_y u^n_{j,k} \rightarrow cos\tfrac{\theta}{2} = \sqrt{1-\eta^2} \tag{23}$$

$$\delta^2_x u^n_{j,k} \rightarrow -4sin^2\frac{\phi}{2} = -4\xi^2, \quad \delta^2_y u^n_{j,k} \rightarrow -4sin^2\tfrac{\theta}{2} = -4\eta^2 \tag{24}$$

and so,

$$\mu_x\delta_x u^n_{j,k} \rightarrow 2i\xi\sqrt{1-\xi^2}, \quad \mu_y\delta_y u^n_{j,k} \rightarrow 2i\eta\sqrt{1-\eta^2}. \tag{25}$$

We also define the amplification matrix G by $\hat{u}^{n+1} = G\hat{u}^n$. With these notations, the frozen coefficient version of (12) is mapped into the Fourier space as follows:

$$\left(I - \frac{2}{3}A\xi^2 + i\sigma\lambda A\xi\sqrt{1-\xi^2}\right)\left(I - \frac{2}{3}B\eta^2 + i\sigma\frac{\lambda B}{I\!R}\eta\sqrt{1-\eta^2}\right)(G-I)$$
$$= -2i\lambda\left[A\xi\sqrt{1-\xi^2}\left(I - \frac{2}{3}\eta^2\right) + \frac{B}{I\!R}\eta\sqrt{1-\eta^2}\left(I - \frac{2}{3}\xi^2\right)\right]. \tag{26}$$

In the general case the matrices A and B do not commute, *thus rendering the analysis* of (26) almost intractable. It is instructive, however, to consider the scalar case. Since the aspect ratio $I\!R = \Delta y/\Delta x$ is arbitrary, and since the original partial differential equation (1), in the scalar linear case, could be transformed to the wave equation

$$u_{\bar{t}} + u_{\bar{x}} + u_{\bar{y}} = 0$$

with $\bar{t} = At, \bar{x} = x$ and $\bar{y} = Ay/B$, we can without loss of generality (in this special linear scalar case) rewrite (26) as:

$$\left(1 - \frac{2}{3}\xi^2 + i\sigma\lambda\xi\sqrt{1-\xi^2}\right)\left(1 - \frac{2}{3}\eta^2 + \frac{i\sigma\lambda\eta}{I\!R}\sqrt{1-\eta^2}\right)(G-1)$$
$$= -2i\lambda\left[\xi\sqrt{1-\xi^2}\left(1 - \frac{2}{3}\eta^2\right) + \frac{1}{I\!R}\eta\sqrt{1-\eta^2}\left(1 - \frac{2}{3}\xi^2\right)\right]. \tag{27}$$

Equation (27) can be rearranged to solve for the amplification factor G :

$$G = \frac{a + i(\frac{\sigma}{2} - 1)b}{a + i\frac{\sigma}{2}b} \tag{28}$$

where

$$a = 1 - \frac{2}{3}\xi^2 - \frac{2}{3}\eta^2 + \frac{4}{9}\xi^2\eta^2 - \frac{1}{I\!R}\sigma^2\lambda^2\xi\eta\sqrt{1-\xi^2}\sqrt{1-\eta^2} \tag{29}$$

and b is the Fourier map of the steady-state operator, i.e.:

$$b = 2\left[\lambda\xi\sqrt{1-\xi^2}(1-\frac{2}{3}\eta^2) + \frac{\lambda\eta}{I\!R}\sqrt{1-\eta^2}(1-\frac{2}{3}\xi^2)\right]. \tag{30}$$

We see immediately from (28) that for $\sigma = 1$ (i.e., the case of second order temporal accuracy) we have a Crank-Nicholson type scheme with $||G|| = 1$. For $\sigma > 1$ (i.e., first order accuracy in time), we have $||G|| < 1$. We have thus demonstrated the unconditional stability of (27) for all values of the cell aspect ratio.

2.4 Stability for the 3-D Case

The three dimensional analog to (27) is

$$(1 - \frac{2}{3}\xi^2 + i\sigma\lambda\xi\sqrt{1-\xi^2})(1 - \frac{2}{3}\eta^2 + i\sigma\frac{\lambda}{I\!R}\eta\sqrt{1-\eta^2})(1 - \frac{2}{3}\varsigma^2 + i\sigma\frac{\lambda}{Q}\varsigma\sqrt{1-\varsigma^2})(G-1) =$$

$$= -2i\lambda\left[\xi\sqrt{1-\xi^2}(1-\frac{2}{3}\eta^2-\frac{2}{3}\varsigma^2) + \frac{1}{I\!R}\eta\sqrt{1-\eta^2}(1-\frac{2}{3}\varsigma^2-\frac{2}{3}\xi^2) + \frac{1}{Q}\varsigma\sqrt{1-\varsigma^2}(1-\frac{2}{3}\xi^2-\frac{2}{3}\eta^2)\right] \tag{31}$$

where ς is the Fourier dual variable defined analogously to ξ and η.

The stability of three dimensional amplification factor G, as defined by (31), is difficult to analyze and we resorted to numerical evaluations of $|G|^2$, using up to 8×10^6 Fourier modes. The numerical study of (31) was carried out on the Cray 2. We found that as in the case of forward Euler approximate factorization second order scheme [4], the amplification factor was conditionally stable. For example for $I\!R = Q = 1$, the stability limit is $\lambda \leq .43$. These stability properties are obtained with the aid of artificial viscosity (AV) term which is of sixth order but still resides on the compact stencil. The AV term is added to the right hand side (explicit term) of (18) and is of the form

$$\mu_{av}\lambda\frac{1}{6!}\delta_x^2\delta_y^2\delta_z^2 \tag{32}$$

where μ_{av} is of order unity. We found $1.5 < \mu_{av} < 2$ to be most efficacious. Without the artificial viscosity term, the allowed value of λ is about one order of magnitude less (e.g., for $R = Q = 1, \lambda$ without using AV is about .035).

3 COMPACT FOUR STEP FOURTH ORDER RUNGE-KUTTA ALGORITHM

The basic four step explicit Runge-Kutta scheme, as proposed in [2], takes the form:

$$
\begin{aligned}
u^0 &= u^n \\
u^{(1)} &= u^{(n)} - \tfrac{1}{4}\Delta t R(u^{(0)}) \\
u^{(2)} &= u^{(n)} - \tfrac{1}{3}\Delta t R(u^{(1)}) \\
u^{(3)} &= u^{(n)} - \tfrac{1}{2}\Delta t R(u^{(2)}) \\
u^{(4)} &= u^{(n)} - \Delta t R(u^{(3)}) \\
u^{(n+1)} &= u^{(4)}
\end{aligned} \tag{33}
$$

where R is the finite difference (or finite volume) representation of the steady operator. For example, in the 2-D second order spatial accuracy case

$$R(u) = \frac{\mu_x\delta_x}{\Delta x} f + \frac{\mu_y\delta_y}{\Delta y} g \tag{34}$$

where f and g are the flux vectors encountered in section 2. If we want fourth order spatial accuracy, then it follows directly that the residual R is modified to read

$$R = \frac{\mu_x\delta_x}{\Delta x}(1 + \frac{1}{6}\delta_y^2) f + \frac{\mu_y\delta_y}{\Delta y}(1 + \frac{1}{6}\delta_x^2) g \tag{35}$$

and we still retain the compact support.

Similarly, in the three dimensional case we have

$$R = \frac{\mu_x\delta_x}{\Delta x}(1 + \frac{1}{6}\delta_y^2 + \frac{1}{6}\delta_z^2) f + \frac{\mu_y\delta_y}{\Delta y}(1 + \frac{1}{6}\delta_z^2 + \frac{1}{6}\delta_x^2) g + \frac{\mu_z\delta_z}{\Delta z}(1 + \frac{1}{6}\delta_x^2 + \frac{1}{6}\delta_y^2) h. \tag{36}$$

Thus (33), with R given either by (35) for the 2-D case or by (36) for the 3-D case, retains all the features of the second order scheme but gains us the fourth order accuracy. In addition one can easily verify by simple analysis that for a given cartesian grid and flow conditions the new fourth order formulation enhances the stability condition. In the 2-D case we have, using (35) rather than (34)

$$\frac{(\Delta t)_{(4)}}{(\Delta t)_{(2)}} \triangleq \frac{(\Delta t)_{4th\ order}}{(\Delta t)_{2nd\ order}} = 1.36. \tag{37}$$

In the 3-D case the gain is even more favorable,

$$\frac{(\Delta t)_4}{(\Delta t)_2} = 1.66. \tag{38}$$

Thus the algorithm efficiency gains are two fold. First, for a given acceptable error level the fourth order accuracy allows a coarser grid, i.e., fewer node points. Second, not only Δt is increased due to the larger cell size but in addition it gains due to (37) (or (38) in the 3-D case).

4 NUMERICAL RESULTS FOR 2-D CASE

The two-dimensional fourth order compact scheme is applied to a shock reflection problem sketched in Figure 1. It shows a 5° shock at Mach 1.95 reflecting from a flat plate. Results are presented both without any explicit artificial viscosity (AV) and with a fourth order AV term of the type $-\frac{\varepsilon}{4!}(\delta_x^2\delta_y^2)$ added to the right hand side. Addition of this AV term reduced the accuracy of the compact scheme to third order (note that in 3-D the sixth order AV terms precludes this reduction in accuracy). Calculations are also made with a second order implicit Euler AF scheme without and with the preceding AV term. Figures 2a and 2b show the results of the compact scheme whereas Figures 3a and 3b show the corresponding results from the second order scheme. It is seen that for $\varepsilon = 0$, both fourth and second order schemes produce very oscillatory results but with $\varepsilon = 0.36$, the results from the compact scheme (Figure 2b) improve dramatically and, in fact, are much better than the corresponding results obtained from the second order scheme (Figure 3b). The shocks captured by the compact scheme are sharper and the convergence is also seen to improve.

Results from a study of grid aspect ratio effect on the compact scheme are also presented here in terms of the similarity parameter κ of reference 1. Three values of κ are considered, namely 1.67, 1.01, and 0.42. Figure 2b corresponds to $\kappa = 1.67$ and figures 4 and 5 correspond to $\kappa = 1.01$ and 0.42, respectively. In all these cases, ε is set equal to 0.36. It is clear from the figures displaying the effect of κ that the best results are obtained for κ near unity. In reference 1, a linear theory (e.g., for weak shock) predicts the same results. It is interesting that we find numerically that this is also the case in the present nonlinear problem.

SUMMARY

1. The steady state solution of the Euler equations of gas dynamics may be achieved to fourth order accuracy using a compact grid stencil of 3×3 and $3 \times 3 \times 3$ in the 2-D and 3-D cases respectively. We presented two examples of such algorithms: one implicit (Euler approximate factorizations scheme) and one explicit (Four-stage Runge-Kutta).

2. Numerical experiments were carried out for the 2-D shock reflection problem, using the implicit algorithm. Comparisons are made with a corresponding second order scheme. The results show that the compact higher order scheme offers marked improvement in both accuracy and convergence rate.

3. In connection with this work we would like to make the following remarks. It is known that the finite difference scheme cannot obtain high order accuracy for conservation-form equations computed on a non-uniform grid. This observation coupled with the ease of obtaining fourth order compact schemes even in 3-D for the Euler equations revives an old debate: Can one use uniform grid and apply conveniently boundary conditions to arbitrary shapes. The potential gain in reduced number of computational nodes and enhanced convergence rate appears large enough to study this question again.

References

[1] R. Shapiro, "Prediction of dispersive errors in numerical solutions of the Euler equations," submitted for presentation at the 2nd International Conference on Hyperbolic Problems, Aachen, West Germany, March 1988.

[2] A. Jameson, W. Schmidt, and E. Turkel, "Numerical solutions of the Euler equations by finite-volume methods using Runge-Kutta time-stepping schemes," AIAA Paper No. 81-1259, June 1981.

[3] D. J. Jones, J. C. South, and E. B. Klunker, "On the numerical solution of elliptic partial differential equations by the method of lines," J. Comp. Physics, Vol. 9, No. 3, June 1982, pp. 496-527.

[4] R. M. Beam and R. F. Warming, "An implicit factored scheme for the compressible Navier-Stokes equations," AIAA J., Vol. 16, 1978, pp. 393-401.

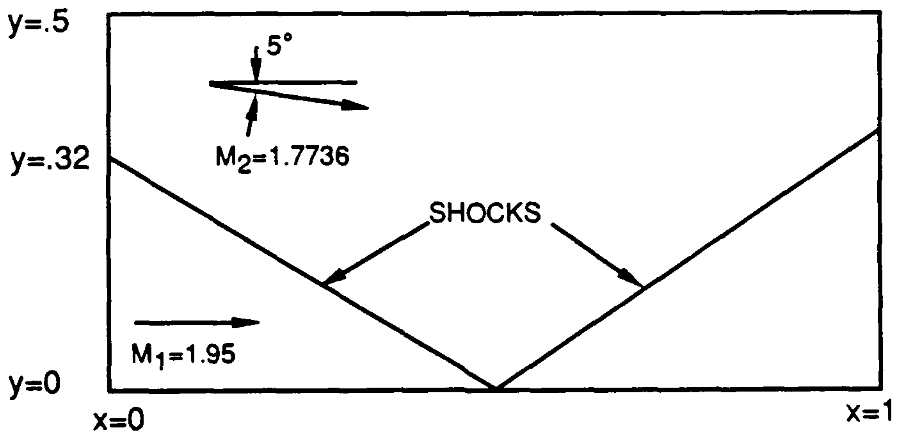


Figure 1: Shock reflection problem

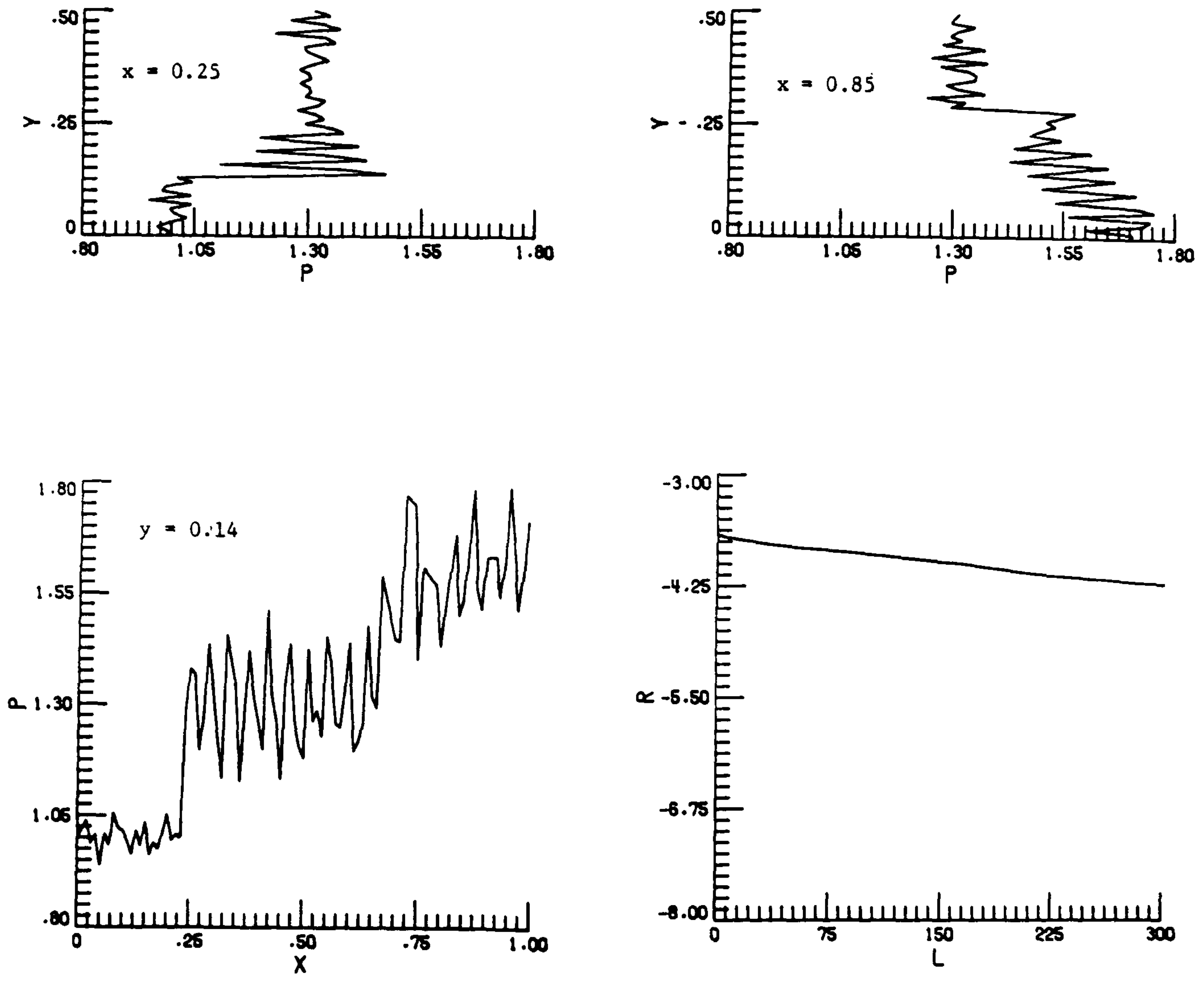


Figure 2a: Pressure distribution and residual plot for compact scheme ($\epsilon = 0, \kappa = 1.67$).

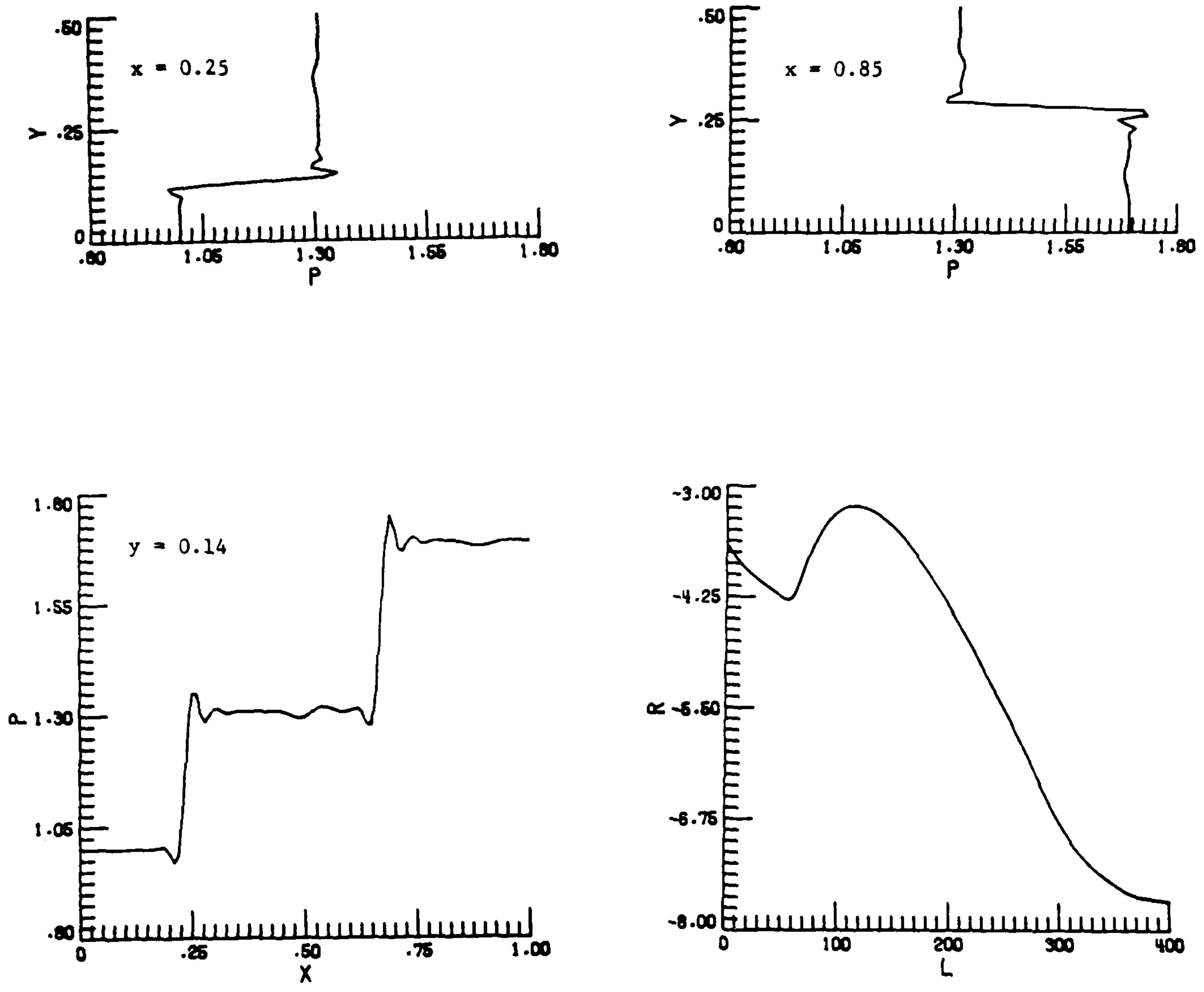


Figure 2b: Pressure distribution and residual plot for compact scheme ($\varepsilon = 0.36, \kappa = 1.67$).

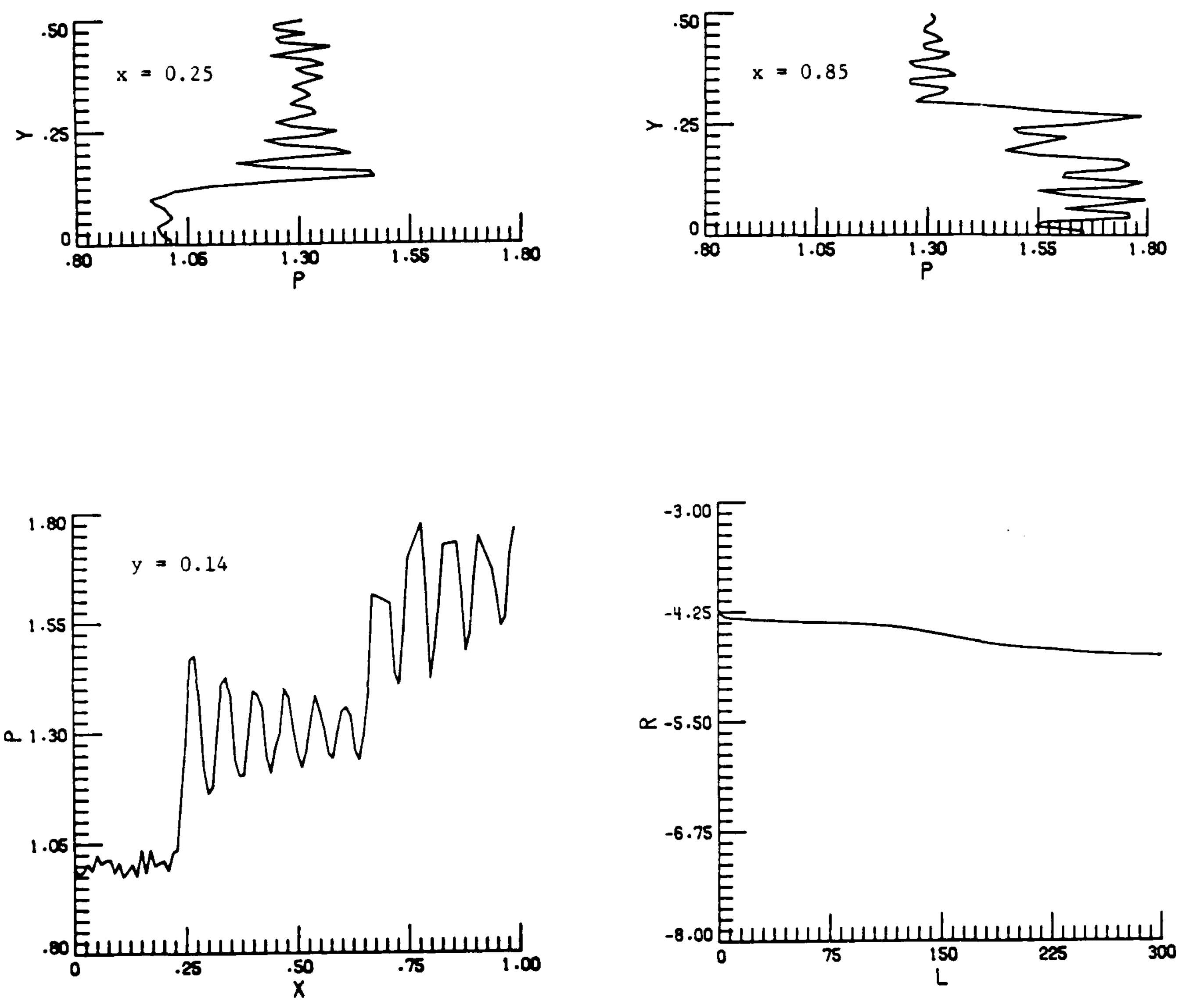


Figure 3a: Pressure distribution and residual plot for second order scheme ($\epsilon = 0, \kappa = 1.67$).

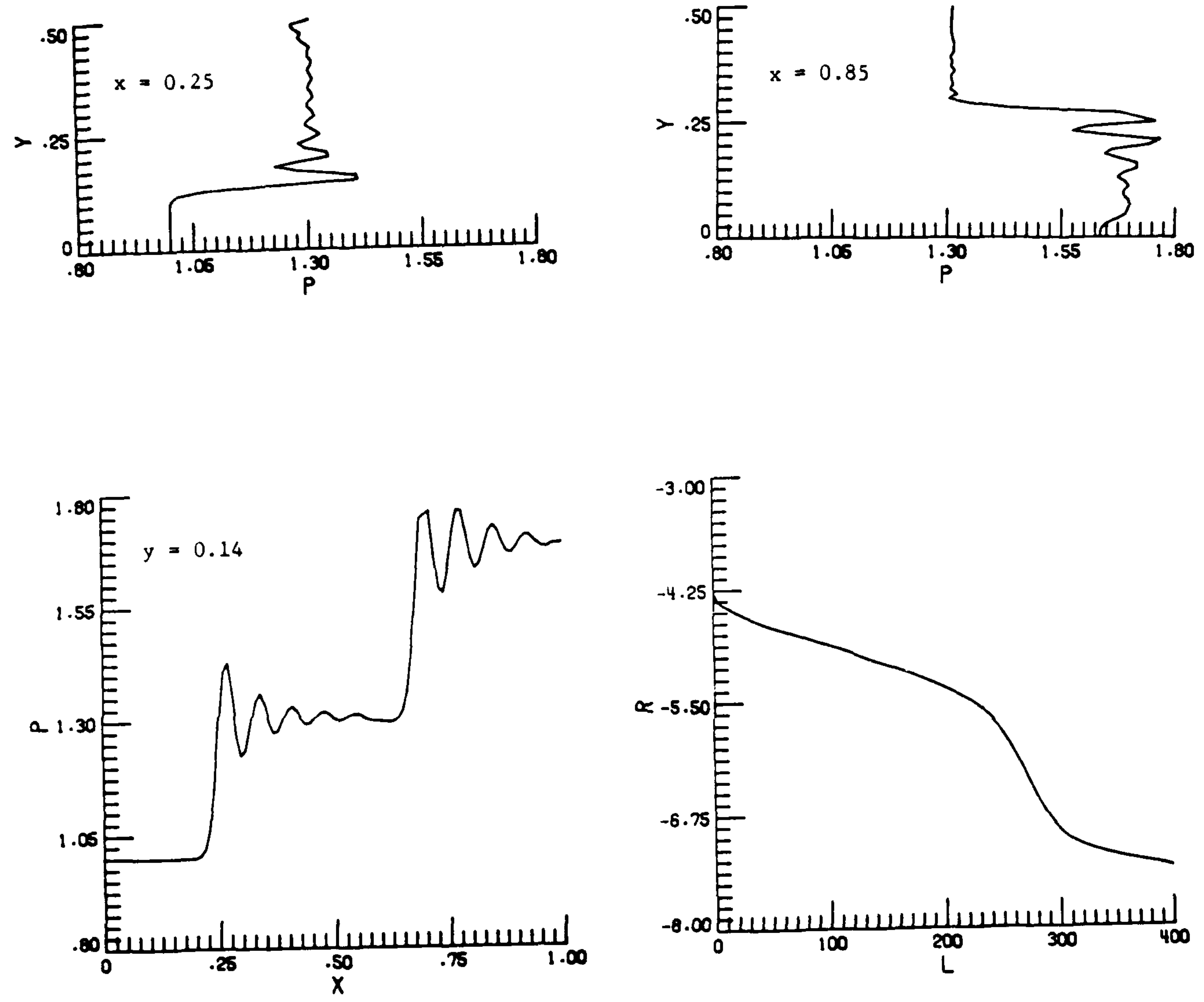


Figure 3b: Pressure distribution and residual plot for second order scheme ($\varepsilon = 0.36, \kappa = 1.67$).

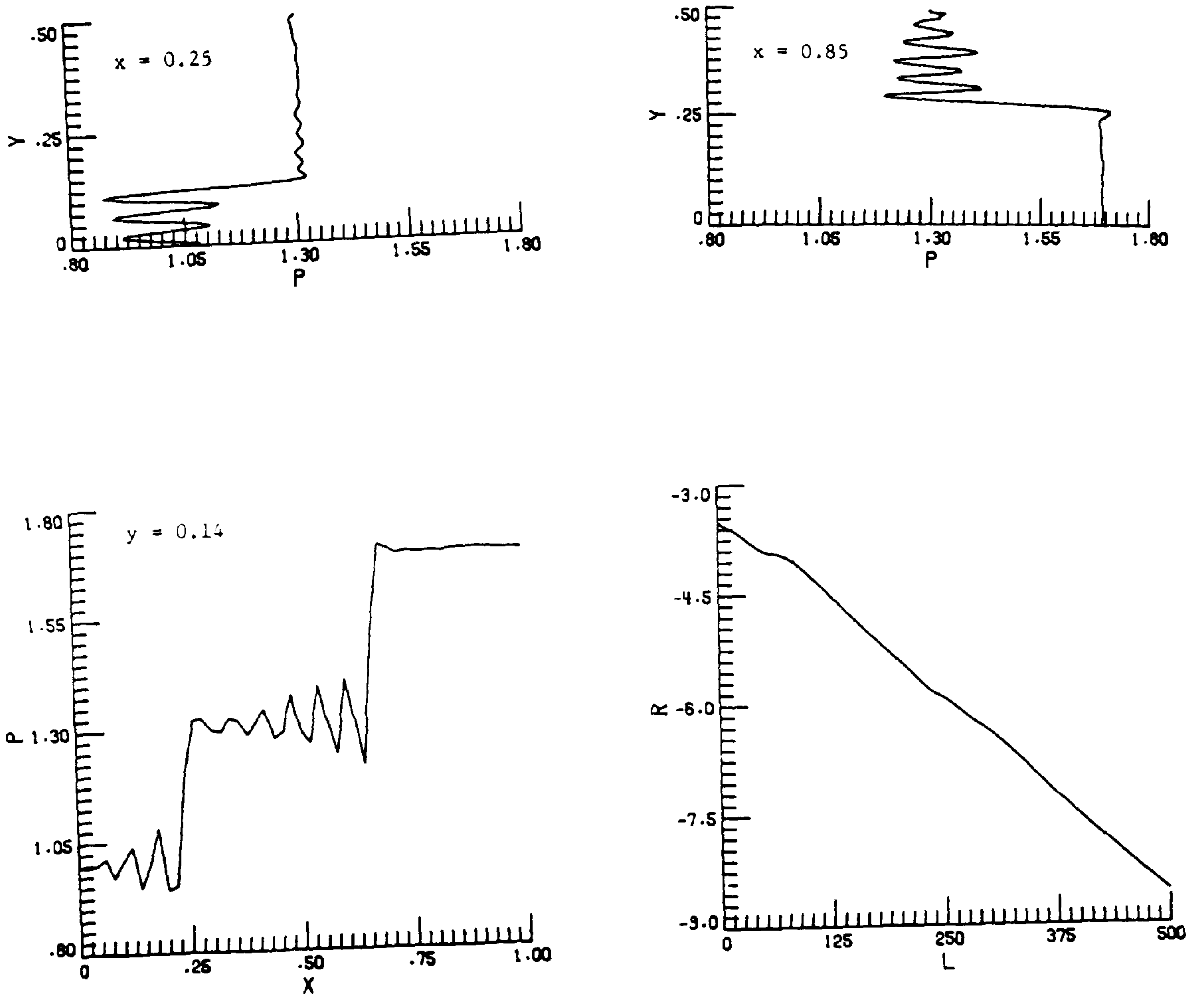


Figure 4: Pressure distribution and residual plot for compact scheme ($\epsilon = 0.36, \kappa = 1.01$).

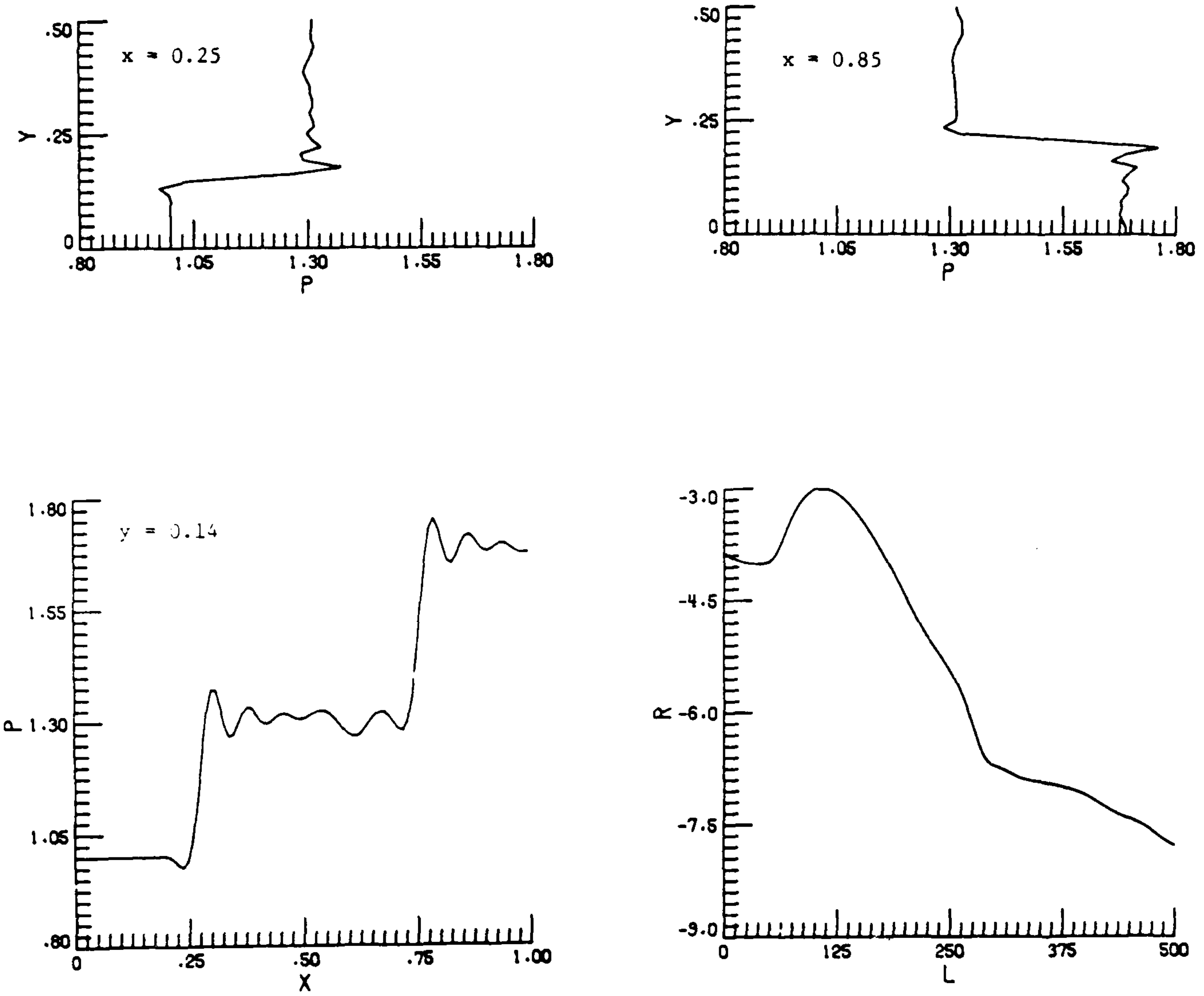


Figure 5: Pressure distribution and residual plot for compact scheme ($\varepsilon = 0.36, \kappa = 0.42$).

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA CR-181625 ICASE Report No. 88-13	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle COMPACT HIGH ORDER SCHEMES FOR THE EULER EQUATIONS		5. Report Date February 1988
		6. Performing Organization Code
7. Author(s) Saul Abarbanel and Ajay Kumar		8. Performing Organization Report No. 88-13
		10. Work Unit No. 505-90-21-01
9. Performing Organization Name and Address Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665-5225		11. Contract or Grant No. NAS1-18107
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code

15. Supplementary Notes
Langley Technical Monitor: Richard W. Barnwell
Submitted to J. of Scientific Computing

Final Report

16. Abstract

An implicit approximate factorization (AF) algorithm is constructed which has the following characteristics.

- In 2-D: The scheme is unconditionally stable, has a 3 × 3 stencil and at steady state has a f urth order spatial accuracy. The temporal evolution is time accurate either to 1st or 2nd order through choice of parameter.
- In 3-D: The scheme has almost the same properties as in 2-D except that it is now only conditionally stable, with the stability condition (the CFL number) being dependent on the "cell aspect ratios," $\Delta y/\Delta x$ and $\Delta z/\Delta x$. The stencil is still compact and fourth order accuracy at steady state is maintained.

Numerical experiments on a 2-D sho :-reflection problem show the expected improvement over lower order schemes, not only in accuracy (measured by the L_2 error) but also in the dispersion.

It is also shown how the same technique is immediately extendable to Runge-Kutta type schemes resulting in improved stability in addition to the enhanced accuracy.

17. Key Words (Suggested by Author(s)) Euler equations, compact schemes, high order accuracy, dispersion	18. Distribution Statement 61 - Computer Prograrning and Software; 64 - Numerical Analysis; Unclassified - unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	16	A02

APPENDIX 6

Prediction of Dispersive Errors in Numerical Solutions of the Euler Equations

by Richard A. Shapiro

CFDL-TR-88-4 March 1988

Presented at the Second International Conference on Hyperbolic Problems, Aachen, Germany, March 14-18, 1988.

Prediction of Dispersive Errors in Numerical Solution of the Euler Equations

Richard A. Shapiro

Computational Fluid Dynamics Laboratory

Department of Aeronautics and Astronautics

Massachusetts Institute of Technology

1 Abstract

Dispersive errors in the discretisation of the steady Euler equations describing the flow of a compressible, inviscid, ideal gas can produce low wave number oscillations near regions of high gradient. A linearised analysis is presented which allows one to predict the location and frequency of these oscillations. This analysis is applied to three numerical schemes: Galerkin finite element, cell-vertex finite element, and central difference finite element. Numerical experiments are presented verifying the analysis. An example showing the applicability of the analysis to a problem with significant nonlinearity is shown.

2 Introduction

Numerical solution of the Euler equations describing the dynamics of an inviscid, compressible, ideal gas are becoming an important tool for the practicing aerodynamicist [1]. In many solutions, low wave number oscillations have been observed in the vicinity of rapid flow variations [2]. These oscillations cannot be explained by problems in artificial viscosity, as their frequency is very low, and the amplitude is relatively independent of the amount of artificial dissipation used.

In this paper, we examine the dispersive properties of three particular algorithms. Each of these algorithms is derived from a finite element formulation, discussed in detail in [3] and briefly discussed in Section 3.1. These three algorithms are the Galerkin finite element method [4,5,6], the cell-vertex finite element method, and the central difference finite element method [7]. The cell-vertex algorithm is identical to the node-based finite volume method [8,9] or the first-order step in Ni's method [10]. On grids with parallelogram elements, the central difference method is equivalent to Jameson's cell-centered finite volume method [11].

The approach taken is to analyse the dispersive properties of the linearised, steady E[illegible] equations on a regular mesh, using the spatial derivative operator for each of three methods discus[illegible] (Galerkin, cell-vertex, central difference). This analysis is applied to a model problem, and the prediction of the frequency and location of the dispersive oscillations is demonstrated. Finally, the analytic theory is validated by comparison with numerical experiments.

3 Solution Algorithm

In this study, the two-dimensional Euler equations describing the flow of an inviscid, compressible fluid are considered. To allow the capture of shocks and other discontinuous phenomena (such as slip lines), the Euler equations are written in conservative vector form as

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho e\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ \rho uh\end{bmatrix}+\frac{\partial}{\partial y}\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ \rho vh\end{bmatrix}=0 \tag{1}$$

or

$$\frac{\partial\vec{U}}{\partial t}+\frac{\partial\vec{F}}{\partial x}+\frac{\partial\vec{G}}{\partial y}=0 \tag{2}$$

where e is total energy, p is pressure, ρ is density, u and v are the x and y flow velocities, $\vec{U}$ is a vector of state variables, $\vec{F}$ and $\vec{G}$ are flux vectors in the x and y directions, and h is the total enthalpy, given by the thermodynamic relation

$$h=e+\frac{p}{\rho}. \tag{3}$$

In addition, one requires the equation of state

$$\frac{p}{\rho}=(\gamma-1)\left[e-\frac{1}{2}(u^2+v^2)\right] \tag{4}$$

where the specific heat ratio γ is taken as a constant (1.4) for all calculations reported.

3.1 Spatial discretization

The finite element approach to discretizing these equations divides the domain into elements determined by some number of nodes (in the current implementation, 4 nodes make up a bilinear element).

Within each element the state vector $U^{(e)}$ and flux vectors $F^{(e)}$ and $G^{(e)}$ are written

$$U^{(e)} = \sum N_i^{(e)}U_i^{(e)} \tag{5}$$

$$F^{(e)} = \sum N_i^{(e)}F_i^{(e)} \tag{6}$$

$$G^{(e)} = \sum N_i^{(e)}G_i^{(e)} \tag{7}$$

where $U_i^{(e)}, F_i^{(e)}$, and $G_i^{(e)}$ are the nodal values of the state vector in element e and the $N_i^{(e)}$ are a set of bilinear interpolation functions on that element. These interpolation functions are expressed in terms of local coordinates (ξ,η), which are related to (x,y) by an isoparametric transformation. Thus, inherent in the formulation that follows are some transformational metrics, which are not shown for clarity.

These expressions can be differentiated to obtain an expression for the derivative in each element in terms of the nodal values (shown here for the state vectors)

$$\frac{\partial U^{(e)}}{\partial x_j}=\sum_{i=1}^{4}\frac{\partial N_i^{(e)}}{\partial x_j}U_i^{(e)}. \tag{8}$$

The flux vector derivatives are calculated the same way.

The expression for the derivatives is substituted into equation (2) and summed over all elements to obtain

$$\frac{\partial\vec{U}_i}{\partial t}=-\frac{\partial\vec{N}}{\partial x}\vec{F}_i-\frac{\partial\vec{N}}{\partial y}\vec{G}_i \tag{9}$$

where $\vec{N}$ is now a global vector of interpolation functions, determined by summing the interpolation functions for each element.

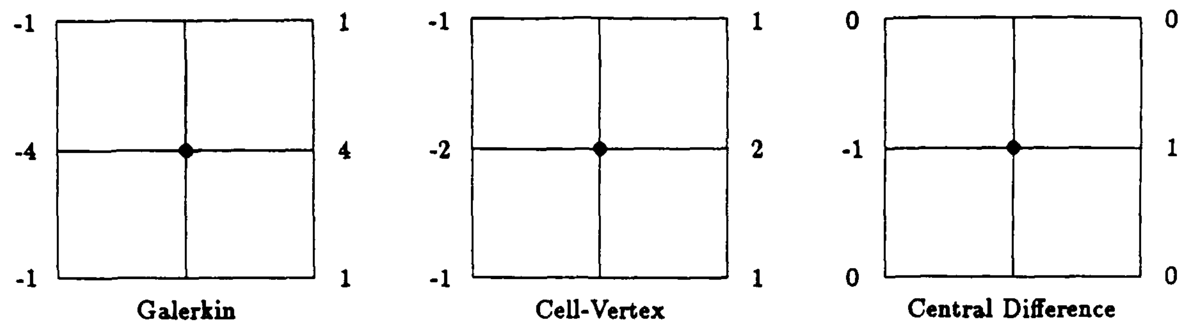


Figure 1: Difference Stencils for x Derivative, Three Methods

The next step can be thought of as a projection onto the space spanned by some other functions N', called test functions, such that the error in the discretization is orthogonal to the space spanned by the test functions (for more detail on the mathematics involved see [12]). To do this, multiply Eq. (9) by $\vec{N}'$ and integrate over the entire domain. This results in the semi-discrete equation

$$M\frac{d\vec{U}_i}{dt} = -\iint (\vec{N}'^T \frac{\partial \vec{N}}{\partial x}\vec{F}_i + \vec{N}'^T \frac{\partial \vec{N}}{\partial y}\vec{G}_i)\, dV \tag{10}$$

$$M = \iint \vec{N}'^T \vec{N} dV \tag{11}$$

or

$$M\frac{d\vec{U}_i}{dt} = -R_x\vec{F}_i - R_y\vec{G}_i \tag{12}$$

where M is the consistent mass matrix and R_x and R_y are what we call residual matrices. The mass matrix M is sparse, symmetric, and positive definite, but not structured, so it is replaced by a lumped (diagonal) mass matrix M_L in which each diagonal entry is the sum of all the elements in the corresponding row of M. This allows Eq. 12 to be solved explicitly. The lumping does not change the steady-state solution, but does modify the time behavior of the algorithm. Finally, this set of ODE's is integrated in time to obtain a steady solution. The details of the time integration, artificial viscosity formulation, and boundary conditions are not critical to the understanding of dispersion, and can be found in [3].

3.2 Choice of Test Functions

Various choices for N' are possible, each giving rise to a particular discretization. If one choses each $N_i'^{(e)}$ to be the corresponding $N_i^{(e)}$, one obtains the Galerkin finite element approximation. If one chooses each $N_i'^{(e)}$ to be a constant, the "cell-vertex" approximation [8,9] results. This approximation is identical to a node-based finite volume method. Finally, if the $N_i'^{(e)}$ are chosen as

$$N'^T = \left[\begin{array}{cccc} \frac{(1-3\xi)(1-3\eta)}{4} & \frac{(1+3\xi)(1-3\eta)}{4} & \frac{(1+3\xi)(1+3\eta)}{4} & \frac{(1-3\xi)(1+3\eta)}{4} \end{array}\right] \tag{13}$$

one obtains the central difference or collocation approximation [7]. On a mesh of parallelograms, this is identical to a cell-based finite volume method. If the mesh is anything else, this equivalence does not hold. Figure 1 shows the difference stencils for an x derivative on a mesh with $\Delta x = 1$. Some properties of these stencils are discussed in the following sections.

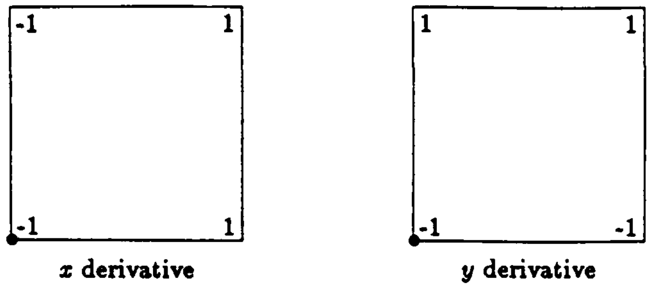


Figure 2: Difference Stencil For a Single Cell-Vertex Element

3.3 Some Properties of the Galerkin Scheme

This approximation has several interesting features. First, it gives the minimum steady-state error, since there is no component of that error in the space of the interpolation functions. Second, for the steady-state Euler equations on a uniform mesh, it is a fourth-order accurate approximation. This interesting feature was pointed out by Abarbanel [13], and can be seen by examining the truncation error for the x and y derivatives. The truncation errors for the derivatives of F and G with respect to x and y can be written (with $\mathcal{D}$ representing the derivative operator) as

$$\mathcal{D}_x F = F_x + 2F_{xxx}\Delta x^2 + 2F_{xyy}\Delta y^2 + \text{H.O.T.} \tag{14}$$

$$\mathcal{D}_y G = G_y + 2G_{yyy}\Delta y^2 + 2G_{xxy}\Delta x^2 + \text{H.O.T.} \tag{15}$$

so the discrete, steady residual $\mathcal{D}_x F + \mathcal{D}_y G$ can be written

$$\mathcal{D}_x F + \mathcal{D}_y G = F_x + G_y + 2\Delta x^2(F_{xxx} + G_{xxy}) + 2\Delta y^2(F_{xyy} + G_{yyy}) + \text{H.O.T.} \tag{16}$$

But in Eq. (16) note that the coefficients of the Δx^2 and Δy^2 terms are the derivatives of the quantity $F_x + G_y$ twice with respect to x and y. For the steady Euler equations, the quantity $F_x + G_y = 0$ so these terms will be higher-order. Thus, on a regular mesh of parallelograms, the Galerkin method should be fourth order accurate. Note that this cancellation of truncation error fails if the equation is inhomogeneous (such as for the Navier-Stokes or Conical Euler equations), or if the mesh is not parallelograms (as in most practical problems).

3.4 Some Properties of the Cell-Vertex Scheme

The most interesting property of the cell-vertex scheme is that it is second-order accurate on any mesh of parallelograms, independent of mesh stretching. To see this, consider the contribution of a single element to the residual at a node. Figure 2 shows the difference stencils for the x and y derivatives at the node denoted by the open circle.

Following the approach of the previous section, the truncation errors for the derivatives of F and G with respect to x and y can be written

$$\mathcal{D}_x F = F_x + F_{xx}\frac{\Delta x}{2} + F_{xy}\frac{\Delta y}{2} + \text{H.O.T.} \tag{17}$$

$$\mathcal{D}_y G = G_y + G_{yy}\frac{\Delta y}{2} + G_{xy}\frac{\Delta x}{2} + \text{H.O.T.} \tag{18}$$

so the discrete, steady residual $\mathcal{D}_x F + \mathcal{D}_y G$ can be written

$$\mathcal{D}_x F + \mathcal{D}_y G = F_x + G_y + \frac{\Delta x}{2}(F_{xx} + G_{xy}) + \frac{\Delta y}{2}(F_{xy} + G_{yy}) + \text{H.O.T.} \tag{19}$$

But again note that in Eq. (19) the coefficients of the Δx and Δy terms are the derivatives of the quantity $F_x + G_y$ with respect to x and y. For the steady Euler equations, the quantity $F_x + G_y = 0$ so these terms will be higher-order. Thus, on any mesh of parallelograms, the cell-vertex method should be second-order accurate. Note that this cancellation of truncation error again fails if the equation is inhomogeneous. This seems to indicate that the cell-vertex method may be superior on highly stretched grids, or on grids with embedded regions. For most practical problems, little difference is observed between the cell-vertex and Galerkin methods.

4 Linearization of the Equations

This section describes the linearisations of the Euler equations, using a method suggested by Giles [14]. The 2-D Euler equations (Eq. 1) can be rewritten

$$\frac{\partial U}{\partial t} + \frac{\partial AU}{\partial x} + \frac{\partial BU}{\partial y} = 0 \tag{20}$$

where

$$U = \begin{bmatrix} \rho \\ u \\ v \\ p \end{bmatrix} \qquad A = \begin{bmatrix} u & \rho & 0 & 0 \\ 0 & u & 0 & \frac{1}{\rho} \\ 0 & 0 & u & 0 \\ 0 & \gamma p & 0 & u \end{bmatrix} \qquad B = \begin{bmatrix} v & 0 & \rho & 0 \\ 0 & v & 0 & 0 \\ 0 & 0 & v & \frac{1}{\rho} \\ 0 & 0 & \gamma p & v \end{bmatrix}. \tag{21}$$

The equations are linearised by "freezing" the A and B matrices, so they can be removed from inside the derivative operator. In the steady state, the time derivative vanishes, so we can write the linearised Euler equations in operator form as

$$(As_x + Bs_y)U = 0 \tag{22}$$

where s_x and s_y are the x and y derivative operators. If we desire non-trivial solutions to this equation, the operator matrix $(As_x + Bs_y)$ must have zero determinant. This is the statement that

$$\begin{vmatrix} us_x + vs_y & \rho s_x & \rho s_y & 0 \\ 0 & us_x + vs_y & 0 & \frac{s_x}{\rho} \\ 0 & 0 & us_x + vs_y & \frac{s_y}{\rho} \\ 0 & \gamma p s_x & \gamma p s_y & us_x + vs_y \end{vmatrix} = 0. \tag{23}$$

Define

$$r = \frac{s_x}{\sqrt{s_x{}^2 + s_y{}^2}} \tag{24}$$

$$s = \frac{s_y}{\sqrt{s_x{}^2 + s_y{}^2}} \tag{25}$$

and Eq. (23) can be expanded to

$$(ru + sv)^2 \left[a^2(r^2 + s^2) - (ru + sv)^2\right] = 0 \tag{26}$$

where a is the speed of sound. This has solutions

$$ru + sv = \begin{cases} 0 \\ \pm a \end{cases}. \tag{27}$$

Now let

$$\vec{s} = \begin{bmatrix} r \\ s \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} u \\ v \end{bmatrix}$$

so that Equation (27) becomes

$$\vec{s} \cdot \vec{u} = \begin{cases} 0 \\ \pm a \end{cases} . \tag{28}$$

Since $\vec{s}$ has unit norm, the non-sero solution will exist only if the flow is supersonic. So far, no restrictions have been placed on the derivative operators s_x and s_y. The analysis above applies to the exact derivative operators as well as any of the discrete operators. The next section introduces the discrete equations and their solution.

5 Fourier Analysis of the Linearized Equations

This section introduces the spatial discretisations of the equations into the linear model, and discusses the consequences of the truncation error in the approximations. Many of the ideas used here can be found in [15], but those analyses were performed for a scalar problem involving only one spatial direction and time.

For purposes of analysis, assume that the equations are discretised on a Cartesian $N_x \times N_y$ mesh with grid spacings in the x and y directions of Δx and Δy. Let $x = j\Delta x$ and $y = k\Delta y$, then assume the state vector is of the form

$$U(j\Delta x, k\Delta y) = \sum_{m=0}^{N_x-1} \sum_{n=0}^{N_y-1} \exp i(j\Delta x \phi_m + k\Delta y \theta_n) U'_{mn} \tag{29}$$

where ϕ_m and θ_n are spatial frequencies in the x and y directions and U'_{mn} is some eigenvector. The spatial frequencies are related to m and n by the relations

$$\phi_m = \frac{2\pi m}{N_x} \tag{30}$$

$$\theta_n = \frac{2\pi n}{N_y}. \tag{31}$$

Now consider a model problem in which $\Delta x = 1$, $\Delta y = \mathcal{R}$ and $v \ll u$, and $u = Ma$. Then Eq. (27) has the solution

$$\frac{s_x}{\sqrt{s_x{}^2 + s_y{}^2}} = \pm \frac{1}{M}. \tag{32}$$

For a particular choice of spatial discretisation, there is a particular dispersive character for a given Mach number M. Table 5 shows s_x and s_y for the Galerkin, cell-vertex and central difference methods, as well as the exact spatial derivative, assuming that ϕ and θ are continuous rather than discrete. Now introduce $s_1 = s_x$ and $s_2 = \mathcal{R} s_y$, square Eq. (32) and solve for s_1/s_2 to obtain

$$\frac{s_1}{s_2} = \mathcal{R}\sqrt{M^2 - 1}. \tag{33}$$

This representation of the dispersion relation has the properties that s_1 and s_2 are functions only of the non-dimensional spatial frequencies ϕ and θ, and all the problem and grid dependent terms are contained in the quantity $\mathcal{R}\sqrt{M^2 - 1}$, which will be called κ. Problems with similar values of κ should have similar dispersive behavior.

One can obtain useful information from these plots of θ vs. ϕ. The slope of a curve on which κ is constant is the spatial "group velocity", or the *angle* at which waves propagate. Waves with large spatial group velocity (the angle on the θ/ϕ plot is close to vertical) will travel at a shallow spatial angle (the wave will move a long way in x for a little change in y). This allows one to predict where the dispersed waves will appear. Figure 4

Method	s_x/i	$\mathcal{R}s_y/i$
Exact Derivative	ϕ	θ
Galerkin	$\frac{1}{3}\sin\phi(2+\cos\theta)$	$\frac{1}{3}\sin\theta(2+\cos\phi)$
Cell-Vertex	$\frac{1}{2}\sin\phi(1+\cos\theta)$	$\frac{1}{2}\sin\theta(1+\cos\phi)$
Central Difference	$\sin\phi$	$\sin\theta$

Table 1: Spatial Derivative Operators for Various Methods

shows the contours of constant κ for the exact spatial derivative operator. Note that the lines of constant κ are straight, indicating that all frequencies travel at the same angle. Moreover, for $\mathcal{R} = 1$, the waves have angle $\tan^{-1}(1/\sqrt{M^2-1}) = \sin^{-1}(1/M)$, which is just the Mach angle. This is expected since we considered the linearised Euler equations.

Figure 5 shows the contour for the Galerkin method. Note that the curves are multiple-valued. For a given ϕ there may be more than one value of θ, or no values of θ. This indicates that there are certain frequencies which do not propagate with real velocities. Practically, the second value is at a high spatial frequency which will be killed off by the artificial damping in the scheme. The other noteworthy thing about Fig. 5 is that the curves depart from the exact Euler solution much later than all the other methods. This is due to the fact that on a uniform, Cartesian mesh, the Galerkin method is fourth-order accurate for the linearised Euler equations. In practice, one never sees this fourth-order accuracy, for three reasons. First, the artificial viscosity introduces some error into the solution scheme. Lindquist has shown [16] that artificial viscosity can have a dominant effect on the solution error. Second, the grids used to solve problems are very seldom Cartesian. On a mesh composed of anything other than congruent parallelograms, fourth-order accuracy can no longer be obtained. Third, the flux calculations for the spatial discretisation introduce some error. These effects combine to make the Galerkin scheme second-order accurate for practical problems.

Figure 6 shows the dispersion plot for the central difference method. Note that the character of the diagram is similar to the Galerkin plot. One would expect the dispersive behavior to be similar to the Galerkin dispersive behavior, and to some extent, this is the case.

Figure 7 shows the dispersion curves for the cell-vertex scheme. Note that the curves are single-valued. Also note that the curvature is opposite the curvature for the Galerkin and central difference methods. For a particular choice of κ, the dispersion curve for the cell-vertex method will lie on the opposite side of the exact dispersion line than the curves for the Galerkin and central difference methods. This implies that the oscillations due to dispersion at a feature (a shock, for example) should appear on the opposite side (ahead or behind) of the feature compared to the Galerkin and central difference oscillations.

An important application of these curves is the prediction of oscillations due to discontinuities such as shocks. In some problems, oscillations before or after a shock can cause the solution algorithm to diverge. For example, in a strong expansion, a post-expansion oscillation may drive the pressure negative, while a pre-expansion oscillation may not be harmful. The dispersion curves allow one to predict the location of these oscillations and choose a solution algorithm which will place them in a safe place. The location of oscillations may be predicted by the following rule: If the θ vs. ϕ curve is concave up, the oscillations will be behind the feature (they travel faster than the exact solution), and if the curve is concave down, the oscillations will be ahead of the feature. For the cell-vertex method, this means that one will see pre-feature oscillations for $\kappa > 1$ and post-feature oscillations for $\kappa < 1$. For the Galerkin and central difference methods, this is reversed: $\kappa < 1$ implies pre-feature oscillations, and $\kappa > 1$ implies post-feature oscillations.

6 Numerical Verification

To verify that the theory above is the correct explanation of the oscillations observed in many problems, several numerical experiments were made. All test problems were for flow over a wedge in a channel with various wedge angles, inflow Mach numbers and mesh aspect ratios. Figure 3 shows the geometry and flow topology for the 10 degree wedge, $M_\infty = 2$ case discussed below. All the calculations were performed on 50x20 grids, and result in similar flow topologies.

The first set of experiments is for a 1/2 degree wedge angle, with $\kappa = 1.732$. Figure 8 show the curves for all three numerical methods and the exact spatial derivatives on a single plot for this value of κ. Here it is apparent that the Galerkin curve stays much closer to the exact curve. Three numerical test cases were run: Mach 2 flow with $\mathcal{R} = 1$; Mach 1.323 flow with $\mathcal{R} = 2$ and Mach 3.606 flow with $\mathcal{R} = 1/2$. A quick examination of the flow geometry gives the physical significance of κ as the ratio of the number of x grid lines crossed by the feature per y grid line crossed. In the Mach 3.606 flow, the shock lies at a much shallower angle, so that for a smaller Δy the same crossing ratio is obtained. A similar argument holds for the Mach 1.323 flow.

Figure 9 shows the Mach number at mid-channel for the central difference method, scaled by the free stream Mach number for the different Mach numbers above. The central difference method is used here because it exhibits the most oscillation with the greatest amplitude. The exact Mach number ratios (M/M_∞) for these shocks are 0.991 for $M_\infty = 2$ and $M_\infty = 3.606$ and 0.986 for $M_\infty = 1.323$. These compare well with the actual data, and explain why two of the curves lie on top of each other. Note that the frequencies of the oscillation are nearly identical. Also note that the frequency changes slightly as one moves further downstream of the shock. This is as predicted by the dispersion curve. As one moves downstream the spatial group velocity increases, meaning ϕ increases slightly. The wavelength predicted by the dispersion relation at $(x, y) = (1.5, 0.5)$ should be about 10.5 points, and the measured wavelength (crest-to-crest) is either 10 or 11 points, depending on where one defines the crest.

The next set of data shows the location of the oscillations for the Mach number 2 case with the three methods. In all the figures shown, the plot is of Mach number at mid-channel. Figure 10 shows the plot for the Galerkin method, Figure 11 for the central difference method and Figure 12 for the cell-vertex method. Note that both the Galerkin and central difference methods exhibit post-shock oscillation, while the cell-vertex exhibits pre-shock oscillation. Also note that the frequency of the Galerkin oscillations is much higher, and with a lower amplitude than the central difference approximation. This is expected since the Galerkin method group velocity errors occur at higher spatial frequencies, (see Fig. 8). As an interesting aside, note that in Fig. 12 the pre-shock oscillations from the reflected shock are visible at the right side of the plot. These figures verify the use of the dispersion curves to predict the location of dispersive phenomena.

The final test cases show the Mach 2 flow over a 10 degree wedge, which generates a shock wave with a normal Mach number of 1.27 and a density ratio of 1.46. This case was chosen because the problem starts to become significantly nonlinear. Figure 13 shows the Mach number on a slice at a 5 degree angle to the x axis for the Galerkin method and Fig. 14 shows the same data for the cell-vertex method. In these cases, κ is about 1.7 ahead of the first shock, and κ is about 0.6 behind the reflected shock (due to the lower Mach number of 1.28 and the changing aspect ratio of the cells). Note that the oscillation positions in Fig. 14 are correctly predicted to be before the first shock, where $\kappa > 1$, and after the second (reflected) shock, where the $\kappa < 1$. Note also that the frequency of the oscillations has increased. This may be due to the tendency of the nonlinear characteristics to point into the shock, tending to make the shock sharper. This self-sharpening behavior has almost completely eliminated the dispersive error in the Galerkin case, but in Fig. 13 small low-frequency oscillations are still visible after the first shock and before the second shock. In many practical applications, the analyst is concerned with the location (pre- or post-shock) more than the frequency, and the linearized analysis is still useful for those applications.

7 Conclusions

The primary conclusion of this study is that the low frequency oscillations sometimes seen near shocks are due to dispersion in the numerical scheme. The linearized analysis presented gives one a method for predicting the location and frequency of these oscillations. The linear analysis is effective in predicting the location of oscillations, even for problems with significant nonlinearity. The central difference finite element method is shown to be inferior to the Galerkin and cell-vertex methods due to its poor dispersive behavior. The Galerkin finite element is shown to be fourth-order accurate for uniform meshes and has the lowest dispersive error. The cell-vertex method is shown to be second-order accurate for any parallelogram mesh and has moderate dispersive error. For the practical analyst, either Galerkin or cell-vertex provides adequate performance.

8 Acknowledgements

This work was supported by the Air Force Office of Scientific Research under contract number AFOSR-87-0218, Dr. James Wilson, Technical Monitor, and by the Fannie and John Herts Foundation. The author wishes to acknowledge the help of Prof. Earll Murman in preparing the report, and of Ellen Mandigo in proofreading the manuscript.

References

[1] A. Jameson, "Successes and Challenges in Computational Aerodynamics," AIAA Paper 87-1184, 1987.

[2] R. Haimes, Personal communication.

[3] R. Shapiro and E. Murman, "Adaptive Finite Element Methods for the Euler Equations," AIAA Paper 88-0034, January 1988.

[4] R. Lõhner, K. Morgan, J. Peraire, and O. C. Zienkiewics, "Finite Element Methods for High Speed Flows," AIAA Paper 85-1531, 1985.

[5] K. Bey, E. Thornton, P. Dechaumphai, and R. Ramakrishnan, "A New Finite Element Approach for Prediction of Aerothermal Loads–Progress in Inviscid Flow Computations," AIAA Paper 85-1533, 1985.

[6] R. Shapiro [illegible]d E. Murman, "Cartesian Grid Finit [illegible] ent Solutions to the Euler Equations," AIAA Paper 87-0559, January 1987.

[7] R. J. Prosan, L. W. Spradley, P. G. Anderson, and M. L. Pearson, "The General Interpolants Method: A Procedure for Generating Numerical Analogs of the Conservation Laws," In *Proceedings of the AIAA Third Computational Fluid Dynamics Conference*, 1977, pp. 106–115.

[8] M. G. Hall, *Cell Vertex Schemes for the Solution of the Euler Equations*, Technical Memo Aero 2029, Royal Aircraft Establishment, March 1985.

[9] K. Powell, *Vortical Solutions of the Conical Euler Equations*, PhD thesis, M.I.T., July 1987.

[10] R. H. Ni, "A Multiple-Grid Scheme for Solving the Euler Equations," *AIAA Journal*, Vol. 20, No. 11, November 1982, pp. 1565–1571.

[11] A. Jameson, W. Schmidt, and E. Turkel, "Numerical Solutions of the Euler Equations by a Finite Volume Method Using Runge-Kutta Time Stepping Schemes," AIAA Paper 81-1259, June 1981.

[12] G. Strang, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, Wellesley, Massachusetts, 1986.

[13] S. Abarbanel and A. Kumar, *Compact Higher-Order Schemes for the Euler Equations*, ICASE Report 88-13, ICASE, February 1988.

[14] M. B. Giles, Personal communication.

[15] R. Vichnevetsky and J. B. Bowles, *Fourier Analysis of Numerical Approximations of Hyperbolic Equations*, SIAM, 1982.

[16] D. R. Lindquist, *A Comparison of Numerical Schemes on Triangular and Rectangular Meshes*, Master's thesis, M.I.T., May 1988.

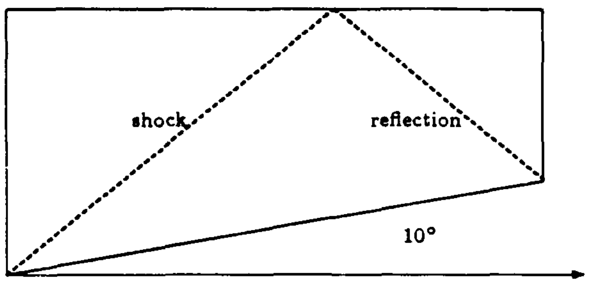


Figure 3: Geometry for 10 Degree Wedge Numerical Test Case, $\mathcal{R} \approx 1$

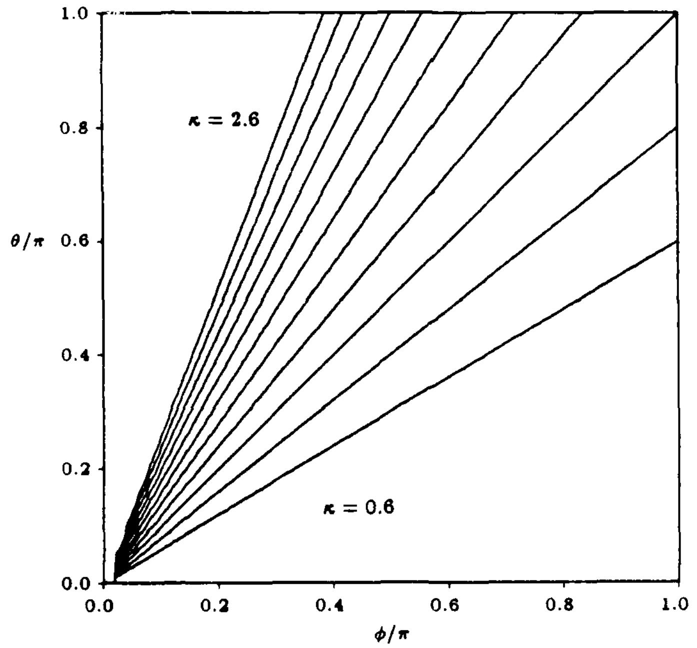


Figure 4: Lines of Constant κ for Exact Spatial Derivatives

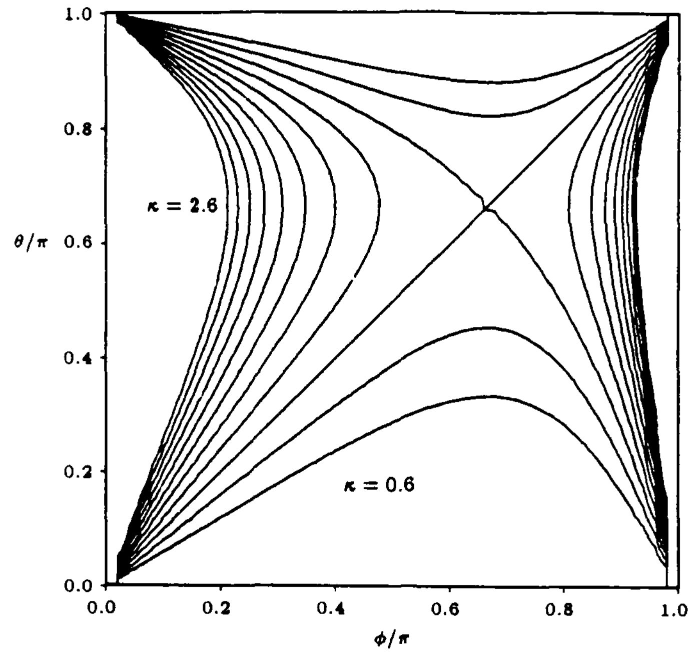


Figure 5: Lines of Constant κ for Galerkin Method

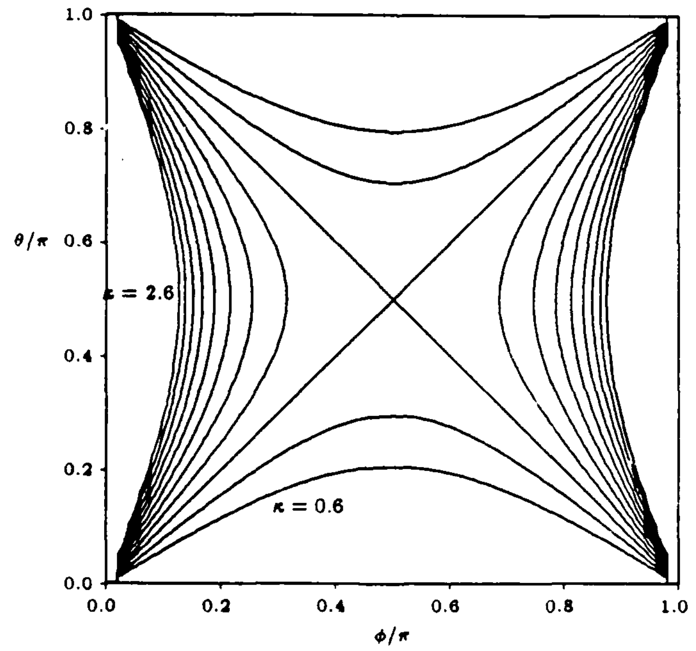


Figure 6: Lines of Constant κ for Central Difference Method

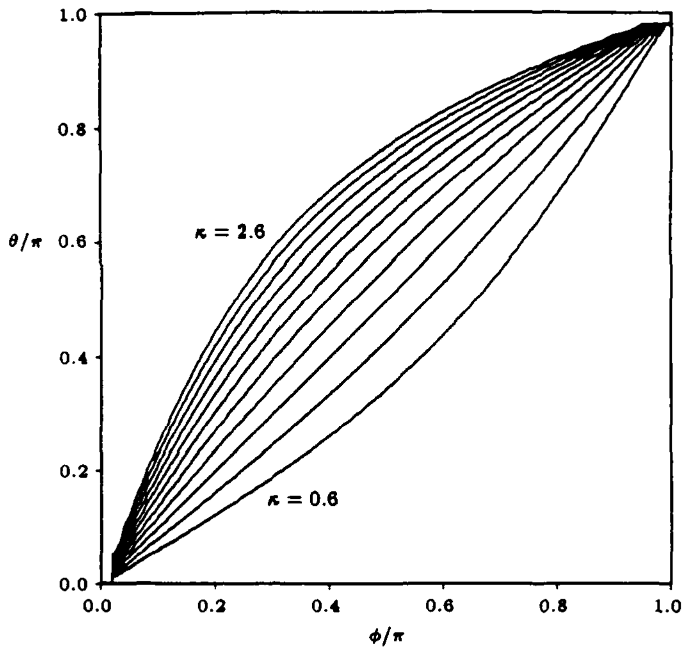


Figure 7: Lines of Constant κ for Cell-Vertex Method

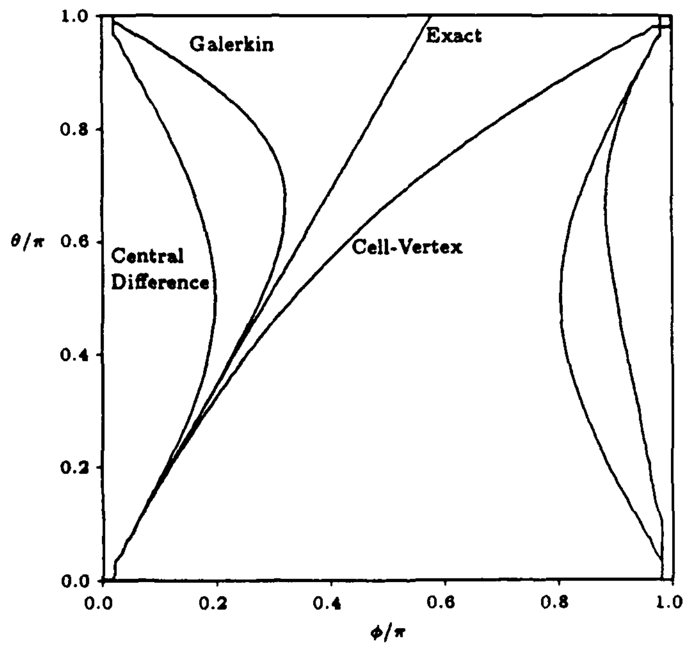


Figure 8: Lines of $\kappa = 1.732$ for All Methods

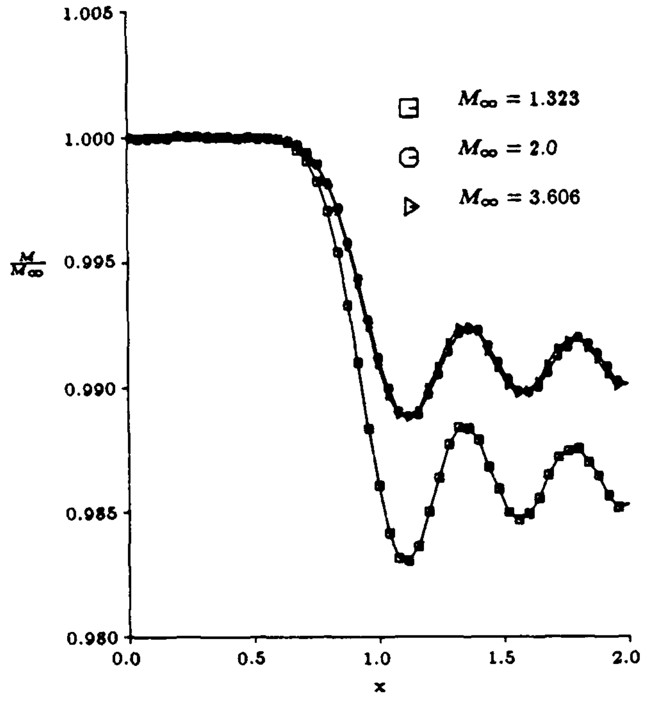


Figure 9: M/M_∞ for $\kappa = 1.732$, Central Difference Method

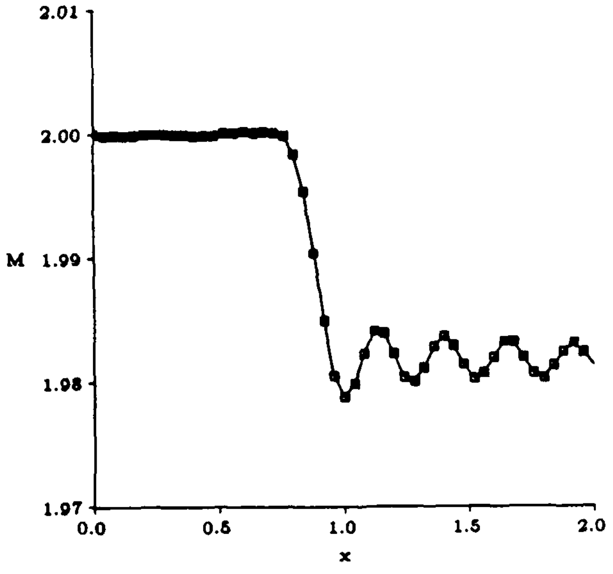


Figure 10: Mid-channel Mach Number, $M_\infty = 2$, 1/2 Degree Wedge, Galerkin Method

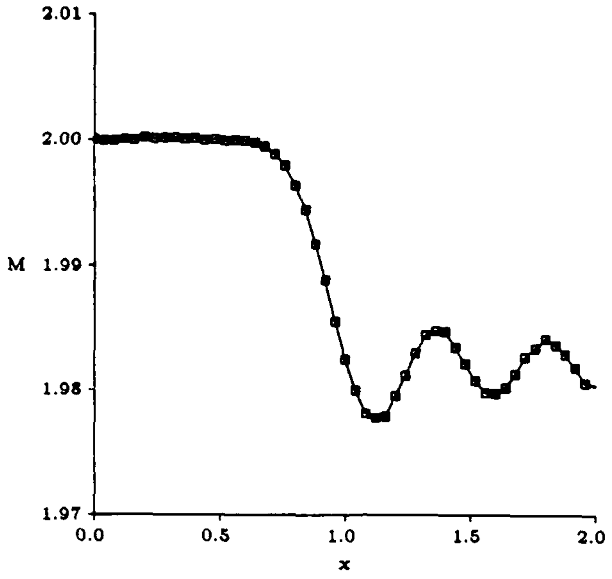


Figure 11: Mid-channel Mach Number, $M_\infty = 2$, 1/2 Degree Wedge, Central Difference Method

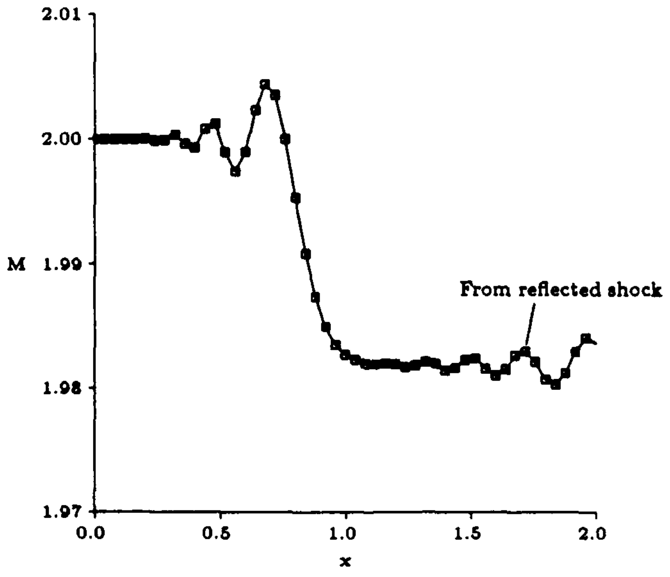


Figure 12: Mid-channel Mach Number, $M_\infty = 2$, 1/2 Degree Wedge, Cell-Vertex Method

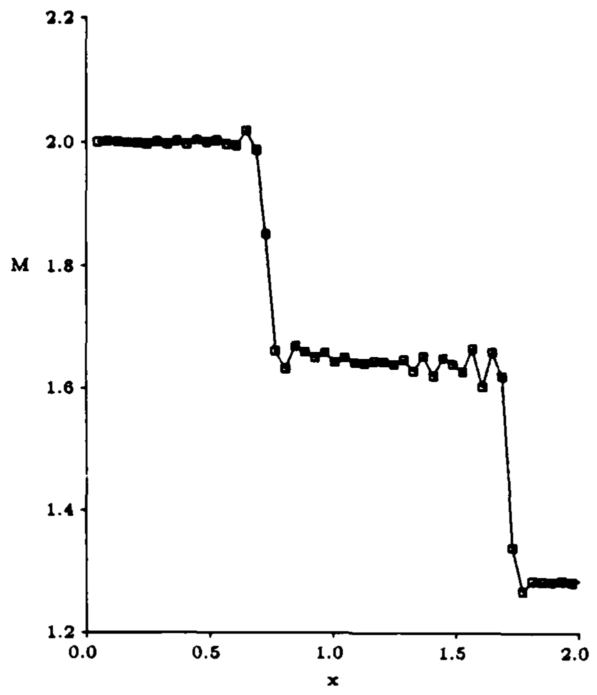


Figure 13: Mid-channel Mach Number, $M_\infty = 2$, 10 Degree Wedge, Galerkin Method

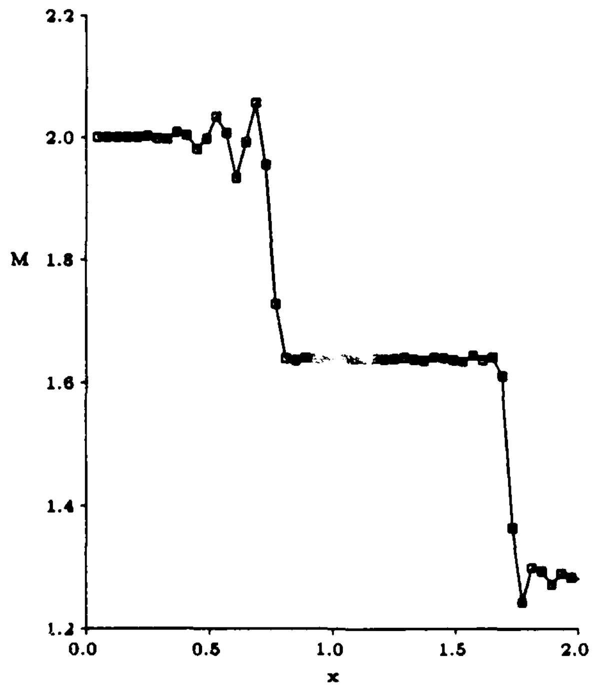


Figure 14: Mid-channel Mach Number, $M_\infty = 2$, 10 Degree Wedge, Cell-Vertex Method

Performance of Several CFD Loops on Parallel Processors

by Earll M. Murman, David Modiano
Robert Haimes, Michael Giles

CFDL-TR-88-3 March 1988

To be presented at the Third International Conference on Supercomputing and Second World Supercomputer Exhibition, Boston, MA, May 15-20, 1988.

PERFORMANCE OF SEVERAL CFD LOOPS ON PARALLEL PROCESSORS

Earll M. Murman, David Modiano,
Robert Haimes, Michael Giles

CFD Laboratory, Department of Aeronautics and Astronautics
Massachusetts Institute of Technology, Cambridge, MA 02139

1 *Abstract*

An implementation of Jameson's popular cell centered finite volume scheme is presented and analyzed for its computational requirements. Timings are given on several computers and compared to LINPACK benchmarks. The program has been analyzed to characterize generic computational constructs. Three simple loops are identified and have been tested on two parallel machines, an ALLIANT FX/8 and an Intel iPSC hypercube. Each loop has different floating point and memory reference features. The most demanding loop, a tridiagonal solver, requires a special algorithm to perform well on the hypercube. It is suggested that the CFD community select some standard loops for use in characterizing the computational workloads inherent in most algorithms.

2 INTRODUCTION

Computational Fluid Dynamics (CFD) involves the numerical solution of systems of nonlinear partial differential equations representing the conservation laws of mass, momentum, and energy applied to a fluid medium. Finite difference, finite volume, or finite element methods are generally utilized to cast the differential equations into a discrete formulation suitable for solution on a computer. Due to the nonlinear nature of the equations, an iterative solution is essential. A variety of approaches are available, but all require large computer memory and many floating point operations to achieve a solution.

CFD is one field which has driven the development of supercomputers. In the future, CFD must exploit the potential gains from parallel processors in order that computational simulations for complex problems can be achieved within realistic times and costs. Many of the algorithms used in CFD are amenable to almost complete parallel computation, yet the benchmarks of large codes often produce disappointing results. Most CFD algorithms bear little resemblance to LINPACK which is commonly used to represent scientific computing workloads. At present there is too little communication between the computer scientists capable of designing high performance parallel computers and the CFD'ers desiring to use such machines. In part, this may be due to the complexities of the CFD algorithms being used. If simpler descriptions were available, the communication and progress might improve.

In this paper, a representative problem from one branch of CFD is examined. The particular problem is the computation of compressible airflow past a transonic transport wing using the Euler equations. A popular explicit finite volume algorithm due to Jameson and Baker [1] as coded by Roberts [2] is considered. Timings are given on several computers for a calculation of the three-dimensional flow past the wing. The code has been analyzed to characterize generic computational structures. From this, three short loops are identified which represent the range of computational tasks embedded in the program. The loops are representative of computational structures in many other CFD codes utilizing structured grid algorithms and therefore should be of general interest. These loops have been tested on two machines representing different architectures and the results examined for parallel efficiencies. One machine, an ALLIANT FX/8, has up to 8 computational elements (CE's) with a shared central memory and a high speed cache. Each CE is a vector processor, and the compiler handles concurrency in one of several ways. The other machine, an Intel iPSC, has up to 32 nodes, each with its own memory, connected in a hypercube topology. The particular model used in this study is an early and not very powerful one, but our main interest was to test parallel efficiencies, not actual MFLOPS.

3 THE ALGORITHM

In this section a brief account is given of the underlying equations and finite volume algorithm for which the timings are given and from which the representative loops are extracted.

3.1 Governing Equations

The three-dimensional unsteady Euler equations in conservation form are:

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho w\\ \rho E\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ \rho uw\\ \rho uH_0\end{bmatrix}+\frac{\partial}{\partial y}\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ \rho vw\\ \rho vH_0\end{bmatrix}+\frac{\partial}{\partial z}\begin{bmatrix}\rho w\\ \rho uw\\ \rho vw\\ \rho w^2+p\\ \rho wH_0\end{bmatrix}=0 \quad (1)$$

with the definition of total enthalpy

$$H_0 = E + \frac{p}{\rho} \quad (2)$$

and the equation of state for a perfect gas

$$p = (\gamma - 1)\left[\rho E - \rho\frac{u^2+v^2+w^2}{2}\right] \quad (3)$$

closing the set. γ is a constant equal to 1.4 for air. The equations are commonly written in vector form:

$$\frac{\partial}{\partial t}\mathbf{U} + \frac{\partial}{\partial x}\mathbf{F}(\mathbf{U}) + \frac{\partial}{\partial y}\mathbf{G}(\mathbf{U}) + \frac{\partial}{\partial z}\mathbf{H}(\mathbf{U}) = 0. \quad (4)$$

U is termed the state vector and represents the 5 dependent variables of the problem. The components of the flux vectors **F**,**G**,**H** along with the pressure p can be evaluated from the components of **U**.

The equations may be integrated over an arbitrary volume V in space bounded by a surface designated as ∂V with an outward normal vector $\hat{n}$ as shown in Figure 1. Using the divergence theorem yields the integral conservation law

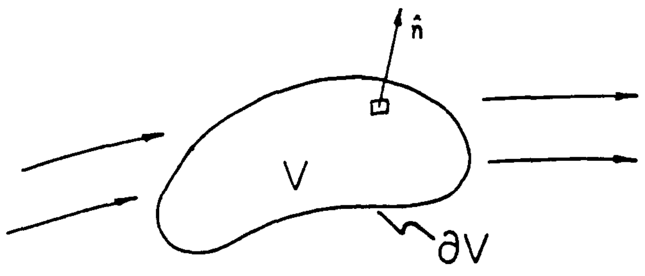


Figure 1: Control volume

form of the equations

$$\frac{\partial}{\partial t}\iiint_V \mathbf{U}\, dV + \iint_{\partial V} (\mathbf{F},\mathbf{G},\mathbf{H})\cdot\hat{\mathbf{n}}\, dA = 0 \quad (5)$$

stating that the time rate of change of mass, momentum, and energy in the volume V is equal to the integrated flux of the mass, momentum, and energy across the boundaries of V. If the integrated fluxes equal zero, the solution is termed "steady". Appropriate boundary conditions must be added to represent the body and flight conditions, but these will not be discussed here.

3.2 Discretized Equations

The basic solution scheme is a finite volume spatial discretization with a multi-stage integration in time as developed by Jameson and Baker [1] which forms the basis of the popular benchmark code FLO57. The particular implementation given here is due to Roberts [2] and the exact coding may somewhat different.

The three-dimensional volume surrounding a wing is filled with a mesh system, illustrated in Figure 2, consisting of $I \times J \times K = N_c$ hexahedral cells. The arbitrary volume

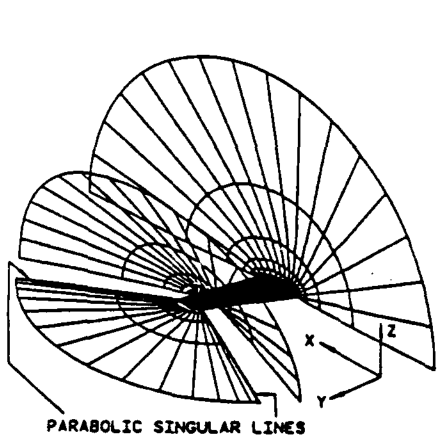


Figure 2: Mesh system

V is then specialized to a typical cell as shown in Figure 3, and the integral equation replaced by the semidiscrete difference equation

$$V_{ijk}\frac{d\mathbf{U}_{ijk}}{dt} = -\sum_{\ell=1}^{6}(\mathbf{F}A_x + \mathbf{G}A_y + \mathbf{H}A_z)_\ell = -\mathbf{R}_{ijk} \quad (6)$$

where A_x, A_y, A_z are the projected areas in the x, y, z directions of each cell face. The flux vectors **F**, **G**, **H** are computed as simple averages between the value in cell i, j, k and its six neighbors $i \pm 1, j \pm 1, k \pm 1$.

The process of computing the residual $\mathbf{R}_{ijk}$ can be done several ways, but the basic procedure in Roberts code is as follows:

1. loop over all cells to compute $p_{i,j,k}$ from the current value of $\mathbf{U}_{ijk}$,
2. set $\mathbf{R}_{ijk} = 0$ for all cells,
3. set up the boundary conditions,
4. loop over all faces along the i direction and compute the products of the following form:

$$(\mathbf{F}_{i,j,k} + \mathbf{F}_{i+1,j,k})A_{x_{i+\frac{1}{2},j,k}} \quad (7)$$

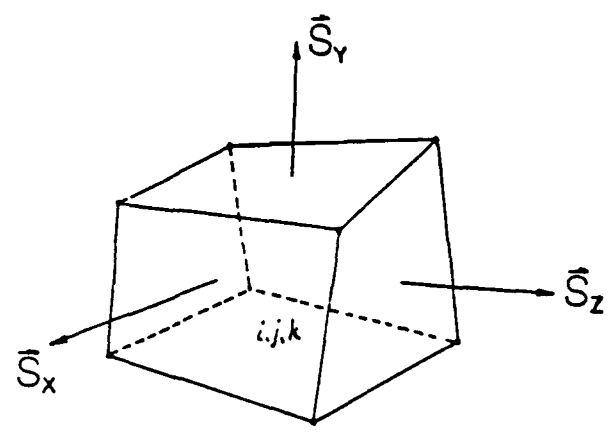


Figure 3: Finite volume cell

with similar terms for $\mathbf{G}A_y$ and $\mathbf{H}A_z$, adding the results to $\mathbf{R}_{i,j,k}$ and subtracting it from $\mathbf{R}_{i+1,j,k}$,

5. repeat the same operations for all faces in the j direction, and then

6. finish with all faces in the k direction, and multiply the final result by $\frac{1}{2}$.

Virtually all CFD algorithms used in the aerospace field have to compute the residual by a similar procedure.

The above discretization requires the addition of smoothing or filter or artificial dissipation terms to damp out high frequency oscillations and to capture shock waves. Typically these smoothing terms involve second and fourth difference operators in the three computational coordinate directions. The details of these operators will not be covered here. The computational structure is similar to the residual calculation given above except that the difference stencil also includes $i \pm 2, j \pm 2, k \pm 2$. Otherwise the process of looping over faces with add and multiply operations is similar. Let the added dissipation operator be denoted by $\mathbf{D}_{ijk}$.

This process creates a large system ($5 \times N_e$) of coupled nonlinear ordinary differential equations for the state vector U. To integrate these equations in time, the multistage scheme of Jameson & Baker [1] is used. Assuming that everything is known at time level n, the values for U at time level $n+1$ are obtained from

$$\begin{aligned}
\mathbf{U}^{(0)} &= \mathbf{U}^n, \\
\mathbf{U}^{(1)} &= \mathbf{U}^{(0)} - \alpha_1 \Delta\tau \left(\mathbf{R}^{(0)} - \mathbf{D}^{(0)}\right), \\
\mathbf{U}^{(2)} &= \mathbf{U}^{(0)} - \alpha_2 \Delta\tau \left(\mathbf{R}^{(1)} - \mathbf{D}^{(0)}\right), \\
\mathbf{U}^{(3)} &= \mathbf{U}^{(0)} - \alpha_3 \Delta\tau \left(\mathbf{R}^{(2)} - \mathbf{D}^{(0)}\right), \\
\mathbf{U}^{(4)} &= \mathbf{U}^{(0)} - \quad \Delta\tau \left(\mathbf{R}^{(3)} - \mathbf{D}^{(0)}\right), \\
\mathbf{U}^{n+1} &= \mathbf{U}^{(4)}
\end{aligned} \tag{8}$$

where the subscripts i, j, k have been omitted for clarity. $\alpha_1, \alpha_2, \alpha_3$ are constants, and $\Delta\tau = \frac{\Delta t}{V}$. V_{ijk} is computed once during initialization and Δt_{ijk} is computed at the beginning of each multistage iteration using a formula from linear stability analysis. For time accurate calculations, Δt is set to the minimum over all cells. The artificial smoothing is evaluated only at the initial stage of the temporal integration to reduce the operation count for the scheme.

Most applications of this algorithm are for problems with a time invariant solution (steady state), and it is advantageous to use various procedures to accelerate the convergence. In essence, time becomes only an iteration parameter and intermediate solutions are of no physical interest. The simplest procedure is to use a different Δt in each cell corresponding to the maximum value allowable from linear stability theory. Another procedure is "enthalpy damping" which adds certain forcing functions to the state vectors at the end of each time step. These forcing functions are zero at convergence. Implicit residual smoothing may also be used by replacing a typical stage of the multistage scheme

$$\mathbf{U}^{(k)} = \mathbf{U}^{(0)} + \alpha_k \delta\mathbf{U} \tag{9}$$

by

$$\mathbf{U}^{(k)} = \mathbf{U}^{(0)} + \alpha_k \delta\mathbf{U}''' \tag{10}$$

where $\delta\mathbf{U}'''$ is computed from

$$\left(1 - \epsilon\delta^2{}_i\right)\left(1 - \epsilon\delta^2{}_j\right)\left(1 - \epsilon\delta^2{}_k\right)\delta\mathbf{U}''' = \delta\mathbf{U}. \tag{11}$$

The δ^2 are second difference operators so that the above equation represents the product of three scalar tridiagonal matrix equations in the three coordinate directions solved by three steps

$$\begin{aligned}
\left(1 - \epsilon\delta^2{}_i\right)\delta\mathbf{U}' &= \delta\mathbf{U}, && (12) \\
\left(1 - \epsilon\delta^2{}_j\right)\delta\mathbf{U}'' &= \delta\mathbf{U}', && (13) \\
\left(1 - \epsilon\delta^2{}_k\right)\delta\mathbf{U}''' &= \delta\mathbf{U}''. && (14)
\end{aligned}$$

Tridiagonal inversions also enter CFD codes for factored implicit algorithms of the Beam-Warming (ADI) class [3], where they may be either scalar or block tridiagonal. Finally, multigrid methods may be employed.

3.3 Computational Resources

Table I shows the floating point operation count and memory references for computing $\mathbf{U}^{n+1}{}_{ijk}$ given $\mathbf{U}^n{}_{ijk}$ for a single cell. It can be seen that about 1500 floating point operations are required including about 60 divide operations. The number of memory references (load and stores) can vary depending upon how a particular machine deals with temporaries. If a given variable is used more than once within a single loop but is retained in a register, then about 1100 load/stores are required. However, if that variable has to be refetched from memory each time it is used,

Subroutine	FLOP	MR	FLOP/MR
Flux Sum	672	440-716	1.53-1.06
Dissipation	432	171-411	2.52-1.05
Pressure	48	24	2
Time Integ	99	113	.88
Enthalpy Damp	32	11	2.91
Res Smoothing	200	335	.60
Total	1483	1094-1610	1.35-.92

Table I - Floating Point Operations (FLOP) and Memory References (MR) per cell per timestep.

then about 1600 load/stores are needed. Since accessing memory is usually a bottleneck, this is an important consideration.

It can be seen that most loops have about one memory reference for each floating point operation. Loops in which the flux vectors or the pressure are computed from the state vector, and the enthalpy damping loop, have 1.5 to 3 times as many floating point operations as memory references. On the other hand, the tridiagonal solvers in the residual smoothing have 50 % more memory references than floating point operations.

For 10^6 cells, a moderate level of resolution, about 2×10^3 time steps are needed to reach convergence requiring a total of 3×10^{11} floating point operations. About 35 32 bit words of memory are needed per cell for a total memory requirement of 14 megabytes.

4 TIMINGS

Table II shows the performance of the above algorithm without the enthalpy damping and residual smoothing routines for several computers. $96 \times 20 \times 20 = 38,400$ cells, a medium level grid, were used for the test case. Halving the grid in each direction to give 307,200 cells would be needed for an truly accurate solution [4]. The calculations were run for 100 time steps to eliminate the set-up overhead. Compiler directives were used as needed on each machine, and all but one loop can be both vectorized and run concurrently without recoding. 32 bit precision is adequate for these calculations, but the actual precision which was used is noted.

For the ALLIANT calculations, good parallel efficiency was achieved up to 5 processors running in the concurrent-outer, vector-inner (COVI) mode. The lower efficiency for 8 CE's could be due to two reasons. Since the outer loop is normally of length 20, using 8 CE's leaves half the CE's idle on the final concurrent cycle. Secondly, the memory access may have saturated with the full 8 CE's running. The timings are for the ALLIANT CE's shipped in 1986.

Machine	Config	Code	MFLOPS	EFF
μVAX II	FPA	Scalar(32)	.2	N/A
FX/8	1 CE	Scalar(32)	.9	N/A
	1 CE	Vector(32)	2.2	1.0
	3 CE	COVI(32)	5.7	.85
	5 CE	COVI(32)	9.6	.86
	8 CE	COVI(32)	11.8	.66
CY 205	2 Pipes	Scalar(64)	8.7	N/A
		VAST II(64)	17.0	N/A
		Vector(64)	48.0	N/A
		Vector(32)	93.0	N/A
XMP/48	1 Proc	Vector(64)	62.0	N/A
IBM 3090	1 Proc	Scalar(32)	10.1	N/A
	1 Proc	Vector(32)	24.8	1.0
	2 Proc	COVI(32)	46.3	.94
	3 Proc	COVI(32)	60.8	.82
	4 Proc	COVI(32)	81.4	.82
	5 Proc	COVI(32)	95.0	.76
	6 Proc	COVI(32)	91.7	.62

Table II - Timings (MFLOPS) and parallel efficiencies (EFF) on various computers.

For the newly announced advanced CE's a 20 % increase in MFLOPS was achieved with 8 ACE's.

The CYBER 205 calculations were run on the John von Neumann center at Princeton University. The VAST II vectorizer produced only about twice the speed of the scalar version. However, a hand-vectorized [5] version which required about 3 months of human time achieved very good performance, particularly with half precision (32 bit). The timing on the NASA Ames CRAY XMP/48 is for a single processor. The code originally written by Roberts was optimized for this machine.

Finally, the IBM timings predate the recently released concurrent compiler for the 3090 and were done by inserting fork/join statements by hand. Some additional temporary variables were defined to avoid multiple divide operations which the compiler didn't optimize out. Good efficiency was achieved up to 5 processors, with the performance for the 6 processor configuration dropping off, presumably due to the same reasons noted for the ALLIANT.

Machine	Config	EULER	LIN(64)	LIN(32)
μVAX II	FPA	.20	.13	.17
FX/8	1 CE	2.2	1.6	1.6
	8 CE	11.8	7.6	7.6
CY 205	2 Pipes	17.0	17.0	
XMP/48	1 Proc	62.0	39.0	
IBM 3090	1 Proc	24.8	12.0	13.0

Table III - Timings (MFLOPS) for present code EULER2 and LINPACK in full (64) and half (32) precision.

Finally, it is interesting to compare the timings for the present code with those of LINPACK[6] in full and half precision as given in Table III. In some cases, the timings are comparable, while in others they can differ by a factor of two. It was assumed that the LINPACK numbers on the CYBER 205 were obtained with the VAST II vectorizer.

5 CFD LOOPS

Three loops have been selected which represent the range of constructs embedded in the algorithm described in Section 3. The loops have been coded and run on the ALLIANT and the Intel iPSC to highlight performance parameters. All calculations have been done with 32 bit words. Listings of the actual FORTRAN code are not included here as it is felt these results are still preliminary and should be discussed within the CFD community before they become adopted as standard test cases.

5.1 Loop 1

The first loop is painfully simple and consists of computing the pressure in all the cells from Equation (2) given the components of the state vector $\mathbf{U} = [\rho, \rho u, \rho v, \rho w, \rho E]^T$. This loop also represents the operations needed to compute the flux vectors $\mathbf{F}, \mathbf{G}, \mathbf{H}$. $\gamma - 1$ is a constant computed and stored at the beginning of the calculation. By defining temporary variables for u, v, w only one divide is needed. For each cell, there are 9 floating point operations, 5 fetches and 1 store. There are no dependencies which inhibit parallel and vector calculation, and the calculation is completely local in that no data is required from neighboring cells. It is a reasonable test of raw computing speed for a given machine.

	$64 \times 64 \times 64$		$32 \times 32 \times 64$	
CE's	MFLOPS	EFF	MFLOPS	EFF
1	2.5	1.0	2.5	1.0
2	4.7	.93	4.7	.93
4	8.6	.86	8.8	.86
8	14.9	.74	15.2	.75

Table IV - Timings for Loop 1 on ALLIANT FX/8.

Table IV shows the results for this loop on the ALLIANT FX/8. It is interesting to note that even for a loop this simple efficiencies significantly lower than unity are realized due either to overhead in controlling 8 different CE's and/or saturation of the memory bus. It is also interesting to see that the overall megaflop performance is close to that realized for the complete code shown in Table II. The timings on the Intel hypercube show parallel efficiencies of unity up to 32 nodes since the loop is local in nature and requires no internodal traffic except for setup. Each node ran at 0.041 MFLOPS.

5.2 Loop 2

The second loop is the flux summation given by Equation (6) for a two space dimension problem. This loop is representative of computational tasks in forming the residual **R** and dissipation **D** vectors given the values of the state vector, the flux vectors, and the projected areas. The loop involves many add-multiply operations which are a commonly occurring construct in many algorithms. The loop has 60 add/subtract, 36 multiplies, 4 stores and 49-100 loads depending upon the ability of the processor to keep temporaries local. A complete description of the loop and the results is given by Modiano [7], and only the results will be reported here.

	Scalar		Vector	
CE's	MFLOPS	EFF	MFLOPS	EFF
1	1.33	1.00	3.47	1.00
2	2.63	.985	6.84	.985
4	5.22	.978	13.1	.948
8	10.2	.949	23.5	.849

Table V - Timings for Loop 2 on ALLIANT FX/8 in scalar and vector mode.

Table V presents the results on the ALLIANT FX/8 for a 64×64 grid. A study was done on 3 CE's with a variable number of grid points and no change in performance was noted until the main memory was exceeded. In scalar mode, very good efficiencies are obtained, while in vector mode some loss of efficiency is experienced with 8 CE's. This may simply be due to saturation of the memory bus. The peak speed obtained of 23.5 MFLOPS represents 25 % of the theoretical maximum of 94.4 MFLOPS. The higher MFLOP numbers for this loop compared to loop 1 are probably due to the absence of divides, the frequency of add-multiply triads which are a vector instruction, and the reuse of variables in cache.

The ALLIANT has a high speed cache between the CE's and the global memory. The global memory is logically divided into blocks corresponding to the size of the cache, and an address location in cache is associated with the corresponding location within each block of the memory. When a load request is made, the cache is first examined to determine if the variable is present. If not, a block of 8 32 bit words including the requested variable is moved to cache. If the next variable requested by the processor is adjacent to the previous one, chances are it is already in cache and does not need to be fetched from main memory (a cache

	NJ				
NK	65	129	257	513	1025
65	8.42	8.25	3.57	2.51	1.62
129	8.47	6.08	3.15	3.12	1.38
257	7.44	5.17	2.50	1.91	1.52
513	7.23	3.07	2.65	2.30	1.64
1025	7.14	5.03	3.24	2.33	

Table VI - Timings for Loop 2 on 3 CE ALLIANT FX/8 with variable indexing of arrays, COVI.

hit). However, one can imagine the worst case of storing data in main memory such that each request overwrites the previously moved data and there are never any cache hits. To illustrate this, the same 64 × 64 problem was done on 3 CE's, but the size of array dimension was varied from 65 to 1025 in each of the j, k indices. Table VI presents the results and illustrates that in some circumstances, a high performance penalty can be paid. Spreading the data out in main memory in the j direction (inner variable) resulted in a sharp decrease in the realized MFLOP performance from 8.42 MFLOPS to as small as 1.38 MFLOPS. This example illustrates the need for application programmers to pay attention to architectural issues and for the manufacturers to to clearly explain the performance principles of each machine.

Loop 2 was run on the Intel iPSC hypercube using domain decomposition. Since the interprocessor communication time is much longer than access time for the local memory on each node, as much data as possible should be kept local. By partitioning the computational domain into subdomains and putting one on each node, only a limited amount of boundary information data needs to be transferred between nodes. A significant amount of syntax must be added to the code to handle initialization and message passing, but the computational strategy is simple.

	32 × 32 = 1024 cells		64 × 64 = 8196 cells	
Nodes	MFLOPS	EFF	MFLOPS	EFF
1	.055	1.00		
2	.055	.995		
4	.054	.986		
8	.053	.968	.055	.996
16	.051	.925	.054	.989
32	.042	.755	.054	.979

Table VII - Timings for Loop 2 on Intel iPSC. MFLOPS is effective performance per node.

Table VII presents the timing results which show that as long as each computational node has enough cells so that the boundary data transmission is minor, excellent parallel efficiency can be achieved. The aggregate MFLOP rate is obtained by multiplying by the number of nodes.

5.3 Loop 3

The third loop which has been tested consists of the three scalar tridiagonal matrix inversions for the implicit residual smoothing given by Equations 11 - 14. A detailed report of the approaches and listing of the FORTRAN programs for doing this on the two processors is given by Haimes, Giles, and Murman [8]. Only an outline of the approach and summary of results will be presented here.

	64 × 64 × 64		32 × 32 × 64	
CE's	MFLOPS	EFF	MFLOPS	EFF
1	1.4	1.0	1.2	1.0
2	2.6	.94	2.3	.96
4	4.5	.83	3.0	.61
8	6.6	.60	4.0	.41

Table VIII - Timings for Loop 3 on ALLIANT FX/8 in vector mode.

For the ALLIANT, the familiar Thomas algorithm, a Gauss elimination procedure for a sparse matrix, was used. A straightforward application with the inner loop being the forward-backward elimination step, leads to dependencies which inhibit both vectorization and concurrency. This can be avoided by making the outer loop the elimination direction and the inner two loops the other directions. Additional storage is then required for temporary variables, but the loop vectorizes and runs parallel. Table VIII presents the results for the ALLIANT. Due to the fact that there is a divide operation and considerable memory traffic (see Table I), the overall performance and efficiency are much less than the other two loops, and there is also a performance penalty for the smaller problem, probably due to poor use of the cache.

For the Intel hypercube, an application of the traditional Thomas procedure will lead to terrible performance as there is no way to organize the data which avoids significant internode traffic in at least two of the three directions. A special algorithm had to be devised which divided each line into subdomains. The forward solve is done in each subdomain, temporarily treating the variables adjacent to the dividing locations as known parameters. This leads to a reduced problem for these variables which can be quickly solved, and the backward substitution is done. The algorithm is fully explained in Haimes et al [8]. About twice the number of operations are required compared to the standard Thomas algorithm, and the coding complexity is much greater, but e⁻ ::ilent parallel efficiency is achieved on a distributed memory architecture. For a 64 × 64 × 64 grid, 51.2 seconds were required on a 32 node configuration, while reducing the number of grid points by a factor of 4 and running on 8 nodes yielded 50.6 seconds.

The tridiagonal solver is the most involved of the three loops, yet is a common algorithm for many applications. The performance on the ALLIANT could undoubtedly be improved by careful blocking of the problem size to stay within the cache size. On the hypercube, good performance can be achieved with a considerable expenditure of programming effort. Both of these lead to the observation that a standard utility should be provided on each machine to free the applications programmer from needing to be an expert on these subjects.

6 UNSTRUCTURED GRIDS

Machine	Config	Code	FEM	LIN(32)
μVAX II	FPA	Scalar	.14	.17
FX/8	1 CE	Vector	1.2	1.6
	3 CE	Vector-Concur.	3.0	
IBM 3090	1 Proc	Vector	7.5	13.0

Table IX - Timings (MFLOPS) for Finite Element Code and LINPACK in half (32) precision.

Currently a number of investigators are developing CFD algorithms using unstructured grids for which i, j, k indexing may not be used. Such approaches either utilize tetrahedra or hexahedra, and often employ adaptative grid approaches wherein cells are subdivided during the iterative solution. Unstructured grids require connectivity matrices to identify which node points are at the vertices of the cells. This "indirect addressing" requires gather-scatter operations which generally lead to a reduced computational throughput. Table IX presents timings on three computers for a finite element method which is similar to the algorithm outlined in Section 3, but which has an unstructured grid. Details of the algorithm are given by Shapiro and Murman [9]. It can be seen that throughput on both the ALLIANT and IBM 3090 are considerably different with the unstructured grids compared to the structured grids. This points to the need for a loop of this variety in order to highlight the features of such computational tasks.

7 CONCLUSIONS

The results reported in this paper are considered as a preliminary step to provide information needed to assist in the design and evaluation of parallel processor computers for CFD applications. A popular CFD algorithm has been analyzed to illustrate the computational requirements, and timings have been compared on several machines and with LINPACK benchmarks. Three loops have been identified which represent typical computational workloads of this algorithm. They also represent generic tasks in most CFD algorithms. The loops have been tested on two parallel processors to illustrate the parallel efficiencies. It would be helpful if the CFD community could decide on several standard loops to assist in the design and evaluation of future parallel computers.

8 ACKNOWLEDGEMENTS

This research was supported by AFOSR Grant 87-0218 monitored by Dr. James McMichael. The assistance of Jerry Kaplan, Larry Fischer and Larry Grossman of ALLIANT Computer Systems Corporation, Mohan Jayaram and Gerry Cocco of IBM, and Ramesh Natarajan and Beth Ducot of MIT in running the benchmarks is appreciated.

References

[1] A. Jameson and T.J. Baker, *Solution of the Euler Equations for Complex Configurations*, AIAA-83-1929CP, July 1983.

[2] T.W. Roberts, *Euler Equation Computations for the Flow Over a Hovering Helicopter Rotor*, PhD thesis, Massachusetts Institute of Technology, November 1986, Also MIT CFDL TR-86-5.

[3] R. Beam and R. Warming, "An Implicit Factored Scheme for the Compressible Navier-Stokes Equations," *AIAA Journal*, Vol. 16, 1978, pp. 393–403.

[4] A. Rizzi, "Three-Dimensional Solutions to the Euler Equations with One Million Grid Points," *AIAA Journal*, Vol. 23, 1985, pp. 1986–1987.

[5] Robert E. Malecki, *Euler Equation Calculations for a Cropped Delta Wing Using the CYBER 205*, Master's thesis, Massachusetts Institute of Technology, August 1986, Also MIT CFDL TR-86-4.

[6] Jack J. Dongarra, *Performance of Various Computers Using Standard Linear Equations Software in a FORTRAN Environment*, Technical Report Tech Memo No. 23, Argonne National Laboratory, October 1987.

[7] David Modiano, *Performance of a Common CFD Loop on Two Parallel Architectures*, Technical Report CFDL-TR-87-11, MIT, 1987.

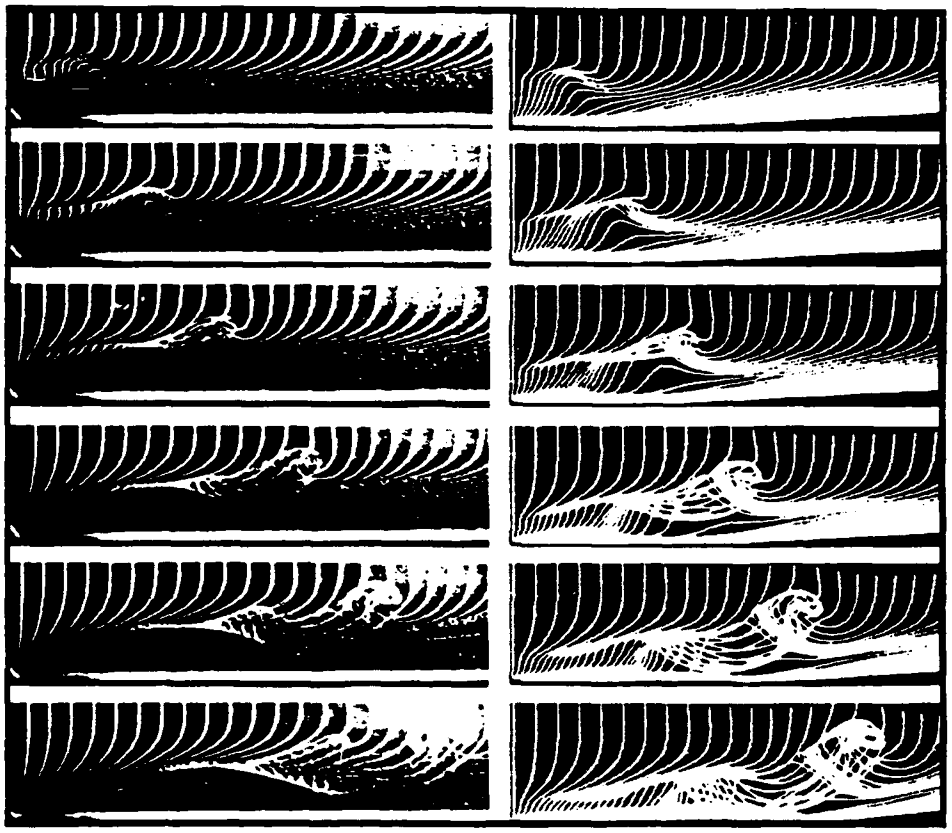
[8] Robert Haimes, Michael Giles, and Earll Murman, *Performance of Implicit Residual Smoothing on Multiprocessor Machines*, Technical Report CFDL-TR-88-2, MIT, February 1988.

[9] R. Shapiro and E. Murman, *Adaptive Finite Element Methods for the Euler Equations*, AIAA-88-0034, January 1988.

APPENDIX 8

Lecture Notes in Physics, Vol 323
Editors: D.L.Dwoyer, M.Y.Hussaini & R.G.Voigt
Springer-Verlag 1989, pp 326-330

11th INTERNATIONAL CONFERENCE ON NUMERICAL METHODS IN FLUID DYNAMICS



The College of William and Mary
Williamsburg, Virginia

June 27 - July 1, 1988

Unsteady and Turbulent Flow using Adaptation Methods

John G. Kallinderis* and Judson R. Baron†
Computational Fluid Dynamics Lab, Dept. of Aeronautics and Astronautics
Massachusetts Institute of Technology, Cambridge, MA 02139

INTRODUCTION

Adaptive grid embedding has proven to be an efficient approach for resolving important flow features. The method was originally applied to inviscid flow involving shocks as the main feature [3,2]. It has recently been extended to include viscous regions with multiple scale phenomena such as shocks and boundary layers [4]. There the *directional embedding* concept was introduced in regions where significant flow gradients exist only in one direction (e.g. boundary layers). *Equation adaptation* also was used to limit the Navier Stokes equations to only appreciably viscous regions, while for the remaining areas the Euler equations were solved. Lastly, a *new finite volume, conservative scheme* was developed to discretize the viscous terms [4], while retaining the inviscid terms discretization due to Ni[6].

The present work extends the previous adaptation methodology and numerical scheme to such complex flows as airfoils in turbulent flow. The detection procedure tracks both flow features and their directionality, and defines *embedding patches* which act as 'filters' to reduce the number of 'randomly' embedded cells. Also a new and general way of implementing an algebraic turbulence model for unstructured grids (quadrilaterals or triangles, the former emphasized here) is described and its accuracy is evaluated. For unsteady simulations, it is shown how one may allow a spatial variation of the time steps while simultaneously maintaining time accuracy. The flow field is divided into temporal levels which coincide with the embedding levels. The accuracy and efficiency of the method is examined for a forced oscillation model problem. The interface treatment is essential for an accurate and robust procedure. Therefore a *Conservative* interface treatment is presented and compared with a nonconservative treatment for steady state problems in both subsonic and supersonic flow.

In the following the concept of *embedding patches*, the Baldwin-Lomax *turbulence model implementation for unstructured meshes*, the method of *spatially varying time steps*, and finally, a *conservative interface treatment*, are all described and evaluated.

EMBEDDING PATCHES

An important consideration when constructing embedded grids relates to the determination of a suitable threshold of the feature detection parameter(s). Additional, extraneous cells may appear near features and are essentially embedded 'noise cells'. Alternatively, cells which should be divided often are overlooked [3]. This characteristic behavior becomes even more severe with directional embedding.

*Research Assistant
†Professor

An *embedded patch* encloses a defined fixed number of cells of the *initial* mesh. It essentially scans the domain and during the scan the included cells are examined for each patch. If the majority of its cells (e.g. 90%) are flagged for division, then *all* cells currently belonging to the patch are flagged for division. Conversely, if very few cells (e.g. 10%) are flagged, none within the patch are embedded. When repeated throughout the domain, the patch procedure can be viewed conceptually as a 'searching window', and acts somewhat like a 'noise filter' which reduces the number of 'randomly' embedded cells.

The illustrative case of a flow around an NACA 0012 airfoil shows the effectiveness of the method. Fig. 5(a) shows Mach number contours just before a *second* level of embedding. The dominant features are a normal shock on the suction side and the upper and lower surface boundary layers. Fig. 5(b) shows the resulting grid. Directional embedding is present above and below the airfoil while the leading edge region is embedded in both directions. The downstream region lying between the two shear layers is not embedded. Comparison of the grid and the resulting solution shows that both the features and their directionality are faithfully captured. The borders between the different regions are fairly free from 'noise' cells.

TURBULENCE MODELING WITH UNSTRUCTURED MESHES

The algebraic model due to Baldwin and Lomax [1] was used as a turbulent flow description. The model implicitly assumes a structured mesh, and its implementation is usually along lines normal to the surface. For an unstructured mesh (quadrilateral or triangular), such normal mesh lines generally do not exist. In the case of quadrilaterals, for example, interfaces interrupt such lines (Fig. 1).

Our approach implements the model in a 'cell-wise' manner. All necessary quantities are calculated at the center of each cell. In this way we avoid using information from outside of the cell, an approach which is common when dealing with unstructured meshes generally. For example, vorticity which is an important quantity for both the inner and outer layer formulation of the model, is calculated using Green's theorem over each cell. Specifically, $\omega = -(1/S_{cell}) \oint_{cell}(udx + vdy)$ where S_{cell} is the cell area. The distance of each cell from the wall is calculated and stored whenever the grid is updated. The only quantities that require information from outside of each cell in order to be evaluated are the Baldwin-Lomax parameters *Fmax* and *Udiff* which are used for the outer layer. These are evaluated by examining all cells associated with each station (cells A,B,C,D,E in Fig. 1). The assumption is that they exhibit no significant streamwise variation over cell E.

The case of a NACA 0012 airfoil in a subsonic turbulent flow tests the accuracy of the method. An initial C-mesh of 33x17 points with two levels of embedding was used for the calculation. In Fig. 4 the pressure coefficient distribution and the lift are compared favorably to experiment [8].

UNSTEADY FLOW WITH SPATIALLY VARYING TIME STEPS

The globally minimum time step required for an unsteady calculation poses a serious limitation to the application of explicit methods. We suggest a procedure which allows a spatial variation of the time steps while simultaneously maintaining time accuracy. To accomplish this the flow field is divided into temporal levels which coincide with the embedding levels (Fig. 2). The same time step is assigned at each level but between two consecutive levels the time steps vary by a factor of two. For the example in Fig. 2, the smaller cells of level 1 are integrated in time twice using a time step δt before the larger cells of level 0 are integrated using a time step $2\delta t$ [5].

Spatial adaptation (cell subdivision to resolve spatial gradients) generally results in *temporal adaptation* (reduction of the time step to resolve temporal gradients) as well [7]. Large temporal gradients may exist in regions where features are present. However, a special algorithm for detection of regions of these temporal gradients is frequently an unnecessary burden since the latter are included in the spatially embedded regions. This results in a significant simplification of the data structure required to handle adaptation.

As a test of the time accuracy of the method, we consider the case of a channel flow with a forced oscillation of the inlet Mach number (Fig. 6). A low Reynolds number ($Re = 10^3$) reduced CPU time. Using one level of embedding the time history of the U-velocity component at a point of the domain was tracked. Free stream flow was used as initial condition. The initial transient decayed after approximately 2 periods. Comparison of the solution with that for a globally embedded mesh shows the agreement between the two histories to be excellent (Fig. 6).

CONSERVATIVE INTERFACE TREATMENT

The existence of embedded regions within the interior of the domain introduces internal boundaries (interfaces) which must be treated carefully. We present a conservative interface treatment and compare it with one that is non-conservative for steady state flows in both supersonic and subsonic cases.

The inviscid terms are evaluated by performing a special line integration at cell E (Fig. 3b) which includes the hanging node b. The cell change in time for the Ni scheme is distributed to all 5 nodes of cell E as shown in Fig. 3a. The viscous terms require piecewise integration of the stresses along the dashed lines (Fig. 3b) on each cell [4]. The interface cell E is divided into five areas which are allocated to each node. The stress fluxes then cancel inside the cell and the treatment becomes conservative. Due to stretching in the cells, there is an error in the viscous terms' evaluation which is locally induced. The distributions to the five nodes due to the smoothing terms (fourth and second order) sum up to zero which makes the smoothing operator conservative. This conservative and a non-conservative treatment [4] have been compared in the following supersonic and subsonic model cases.

Interfaces near a Shock. Channel flow with a $M = 1.35$ shock was used as a test with embedding near the shock (Fig. 8). The embedded region is enclosed by the crosses. In Fig. 9 we compare mass flow across the channel for the two treatments and for a globally fine mesh without interfaces. The nonconservative treatment clearly induces a mass flow error, while the conservative one reproduces the same result as the globally fine.

Interfaces inside a Boundary Layer. The same geometry in a subsonic flow ($M = 0.5$) and with embedding at midheight inside the boundary layer was used to test the conservative and nonconservative treatments. Fig. 7 shows skin friction distributions for both the embedded and the globally fine grids. In this case the non-conservative interface treatment provides excellent agreement with the globally fine solution. An interpretation is that the stretching error in the conservative treatment is more important than the non-conservation error.

CONCLUSIONS

Concepts for adaptive procedures have been examined. Feature detection is made more accurate by the introduction of embedded patches which act as 'noise filters' in the embedding procedure. A method to implement the Baldwin-Lomax algebraic turbulence model when applied to unstructured meshes proves to be accurate. Spatially varying time steps based on embedding zones can be used for unsteady problems leading to accurate results with reduced CPU times. Lastly, a conservative interface treatment is important in regions where shocks exist, but not necessarily inside boundary layers where the stretching error in the evaluation of the viscous terms may be dominant.

References

[1] B. S. Baldwin and H. Lomax. AIAA Paper 78-257, 1978.

[2] M. Berger and A. Jameson. *AIAA Journal*, 23:561–568, Apr 1985.

[3] J. F. Dannenhoffer III and J. R. Baron. AIAA Paper 85-0484, 1985.

[4] J. G. Kallinderis and J. R. Baron. AIAA Paper 87-1167-CP, 1987.

[5] R. Lohner, K. Morgan, J. Peraire, and O. C. Zienkiewicz. AIAA Paper 85-1531, 1985.

[6] R.-H. Ni. *AIAA Journal*, 20:1565–1571, Nov 1982.

[7] M. Pervaiz and J. R. Baron. *Comm. in Applied Numerical Methods*, 4(1):97–111, Jan 1988.

[8] J. J. Thibert, M. Granjacques, and L. H. Ohman. Technical Report AGARD AR 138, 1979.

Acknowledgement *Sponsored by the Air Force Office of Scientific Research, Contract Monitor Dr. James McMichael.*

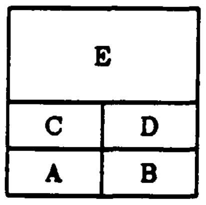


Figure 1: Cell group at a Station

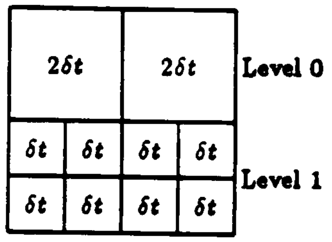


Figure 2: Spatially varying time steps

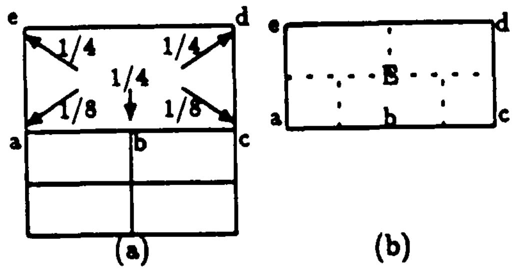
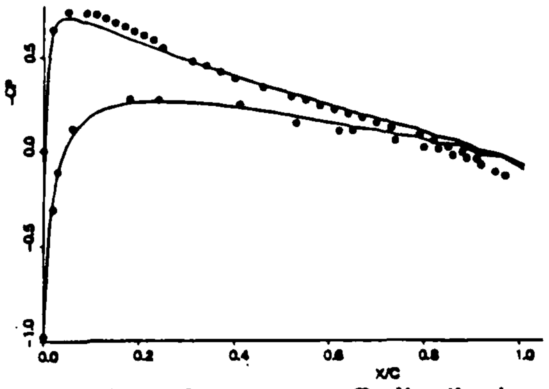
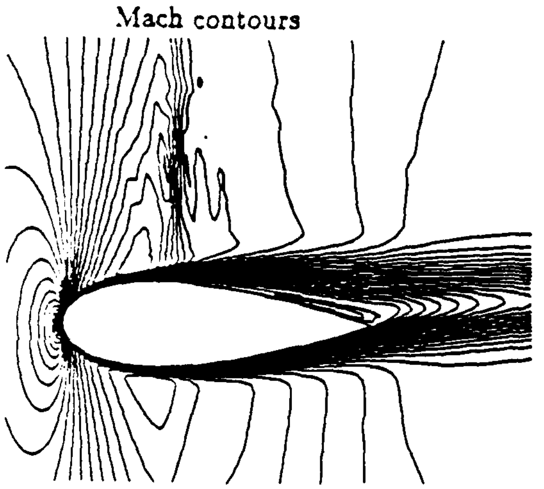


Figure 3: Conservative Interface Treatment

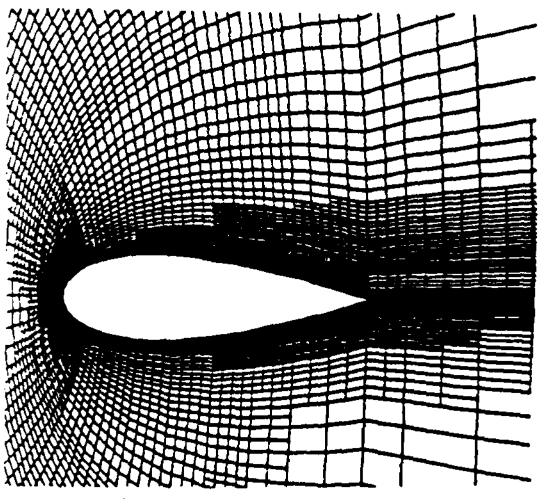


Comparison of pressure coeff. distributions
— present work ($Cl = 0.197$)
o Experiment [8] ($Cl = 0.195$)

Figure 4: NACA 0012 ($M_\infty = 0.5, Re = 2.91 \times 10^6$)



(a) solution before embedding
(vertical scale enlarged)



(b) resulting grid
(vertical scale enlarged)

Figure 5: NACA 0012 ($M_\infty = 0.8, Re = 10^5, \alpha = 2°$)

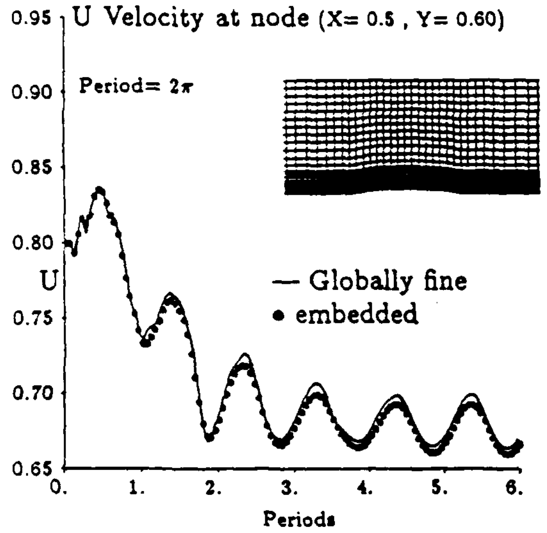


Figure 6: Time history ($M_\infty = 0.8 + 0.04sint$)

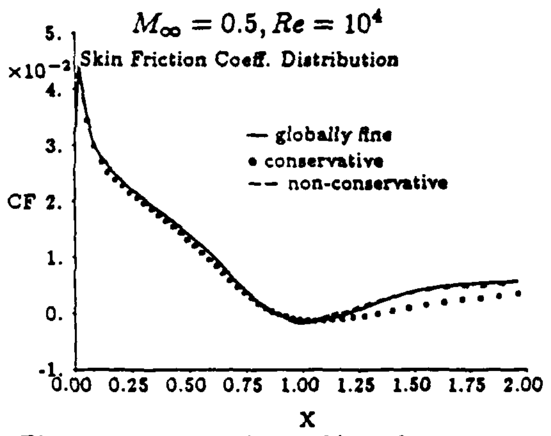


Figure 7: Comparison of interface treatments

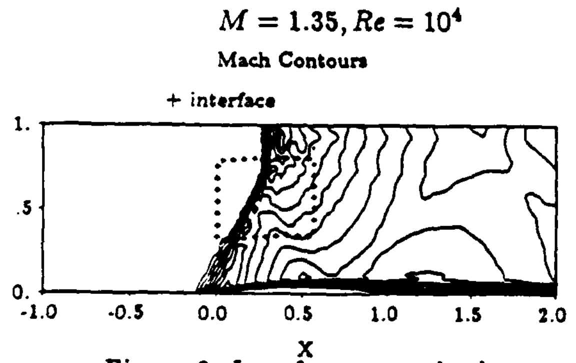


Figure 8: Interfaces near shock

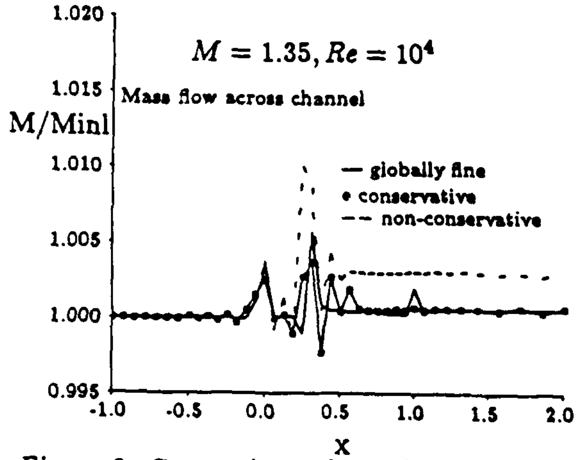


Figure 9: Comparison of interface treatments

PRELIMINARY SUMMARY OF TURBULENT SPOT EXPERIMENT

Three stages of transition have been observed and reported on earlier. In the first stage the wave packet can be treated as a superposition of all two- and three-dimensional waves according to linear stability theory, and most of the energy then is centered around a wave corresponding to the one that is most amplified. In the second stage, most of the energy is transferred to waves which are centered around the wave having half the frequency of the most amplified linear mode. During this stage the amplitude of the wave packet increases from 0.5% to 5% of the free stream velocity (U_o). In the final stage a turbulent spot develops, and the amplitude of the disturbance increases to 27% of U_o.

The transition process that was obtained in the present experiment is demonstrated in Figure 1. On the left of the figure the time evolution of the velocity fluctuations in the stream direction are shown; on the right their associated power spectra are plotted versus the nondimensional frequency β. These were measured along the centerline and inside the boundary layer where the mean velocity is 35% of the free stream velocity. The results are shown for three down-stream stations (XD) representing the three stages of transition mentioned above. Thus, increasing the initial amplitude of the wave-packet did not result in any substantial change of the transition process except for the fact that the entire process was advanced further upstream. The linear stage covers the down-stream distance where nonlinear interactions between various modes of the wave-packet are insignificant. Therefore, increasing the initial amplitude level of the wave-packet results in shortening of the linear stage. Consequently the subharmonic and the spot formation stages have been accomplished earlier at X = 230 cm and 300 cm respectively, relative to X = 280 cm and X = 350 cm respectively in previous experiments.

The corresponding 2-D spectra are shown in Figure 2. They were obtained by measuring the u and w velocity components in the X-Z plane. These 2-D spectra indicate again that when the initial amplitude level of the wave-packet is increased, its temporal-spatial structure remains unchanged through the various stages of transition except for a backwards shift in the X direction. However, the dimensional components of the wave-packet do evolve differently. For example, the dimensional frequencies of the waves which are undergoing resonance in the subharmonic stage are different between the two cases and do depend on the initial amplitude level.

For 3-dimensional disturbances the linearized equations of motion reduce to a set of two equations: (see also Benney and Gustavsson, Studies in Applied Mathematica <u>64</u>:105-209, 1981)

$$\left\{ D^4 - 2kD^2 + k^4 - i\,\mathrm{Re}\,\alpha_x \left[\left(\bar{U} - \frac{\beta}{\alpha_x} \right) (D^2 - k^2) - \bar{U}'' \right] \right\} v = 0 \qquad (1)$$

$$\left\{ D^2 - k^2 - i\,\alpha_x \mathrm{Re} \left[\bar{U} - \frac{\beta}{\alpha_x} \right] \right\} \eta = i\,\alpha_z\,\mathrm{Re}\,\bar{U}'\,v \qquad (2)$$

$$u = \frac{i}{k^2} (\alpha_x Dv - \alpha_z \eta)$$

$$w = \frac{i}{k^2} (\alpha_z Dv + \alpha_x \eta) \qquad (3)$$

where

$$k^2 = \alpha_x^2 + \alpha_z^2$$

$$D \equiv d/dy$$

Equation 1 is the familiar homogeneous Orr-Sommerfeld equation, for the vertical velocity v. The equation for the vertical vorticity, η, is inhomogeneous, and its homogeneous part has a different eigenvalue operator. While the structure of v depends only on the solution of the Orr-Sommerfeld equation, the structure of u and w depends on both equations. Therefore, one must be careful about the interpretation of Squire's theorem. In other words, it is correct that the 2-dimensional wave is the one that is most amplified, and that it is the most energetic wave of the v velocity component. However, it is not necessarily true that the 2-dimensional mode is the most energetic wave of the u velocity component. This point is demonstrated in Figure 3. The 2-D power spectra measured at X = 170 cm (linear stage) and at a Y location where U/U_0 is 0.5 are shown. At the top of the figure the 2-D power-spectra of the u and v velocity components measured with X-wire are shown. At the bottom of the figure the 2-D power-spectra of the u and w velocity components measured with V-shaped wire are shown. It is evident that while the most energetic mode of the v velocity component is a 2-dimensional wave, the most energetic mode of the u velocity component is in fact a 3-dimensional wave.

At three downstream stations representing the linear, subharmonic and spot formation stages, measurements of all 3 velocity components were made, mapping the entire Y-Z plane. At each height Y from the plate a double Fourier transform in time and spanwise direction was made, and therefore the Y distribution for each mode was obtained. The distributions for the most amplified two-dimensional wave and its three-dimensional subharmonic wave which has the same phase velocity are shown in Figure 4 for the three downstream locations. The profiles of the streamwise and vertical velocity fluctuations for the two-dimensional wave are plotted on the left, while the profiles corresponding to the three-dimensional subharmonic wave are shown on the right. The triangular symbols, representing the calculated data points, are compared with the solutions of the linear stability theory given by the solid lines. The theoretical calculations are compared with experiments by equating the areas under each curve of the streamwise fluctuation only. The agreement between the theoretical calculations and the data is fairly good for the linear and subharmonic stages, but is poor for the spot formation stage.

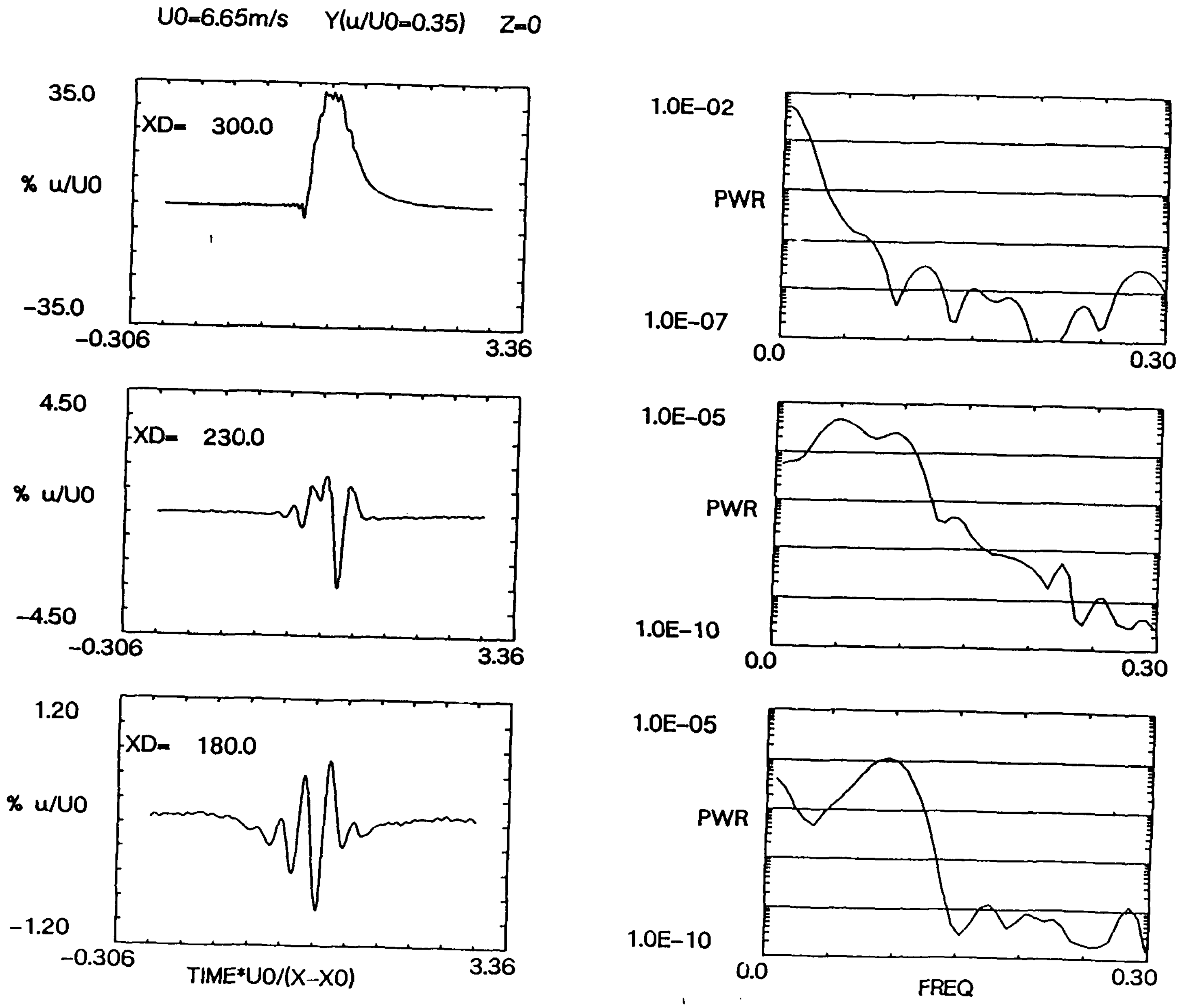

U0=6.65m/s Y(u/U0=0.35) Z=0
35.0
XD= 300.0
% u/U0
-35.0
-0.306
3.36
4.50
XD= 230.0
% u/U0
-4.50
-0.306
3.36
1.20
XD= 180.0
% u/U0
-1.20
-0.306
TIME*U0/(X-X0)
3.36
1.0E-02
PWR
1.0E-07
0.0
0.30
1.0E-05
PWR
1.0E-10
0.0
0.30
1.0E-05
PWR
1.0E-10
0.0
FREQ
0.30

FIGURE 1

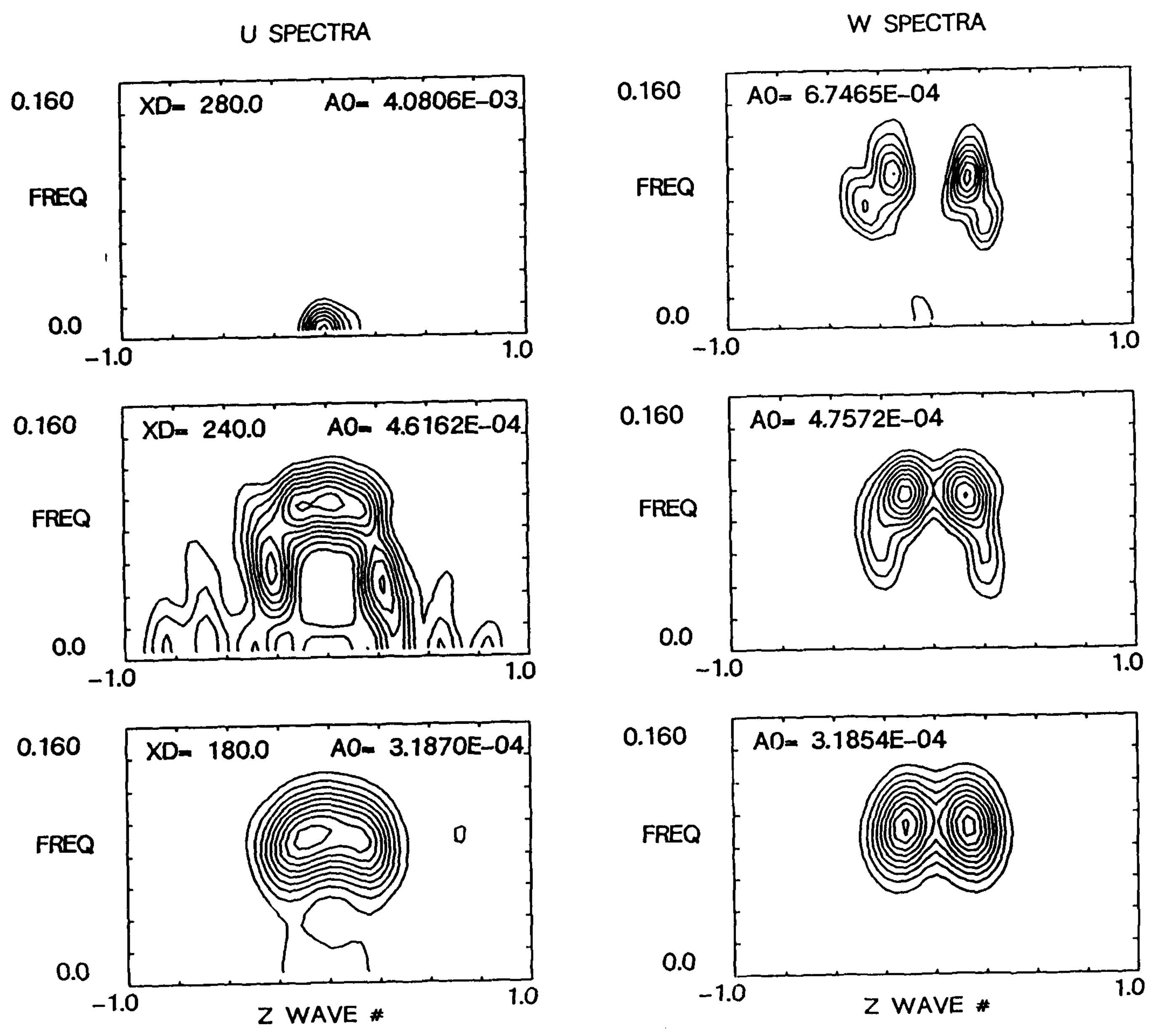
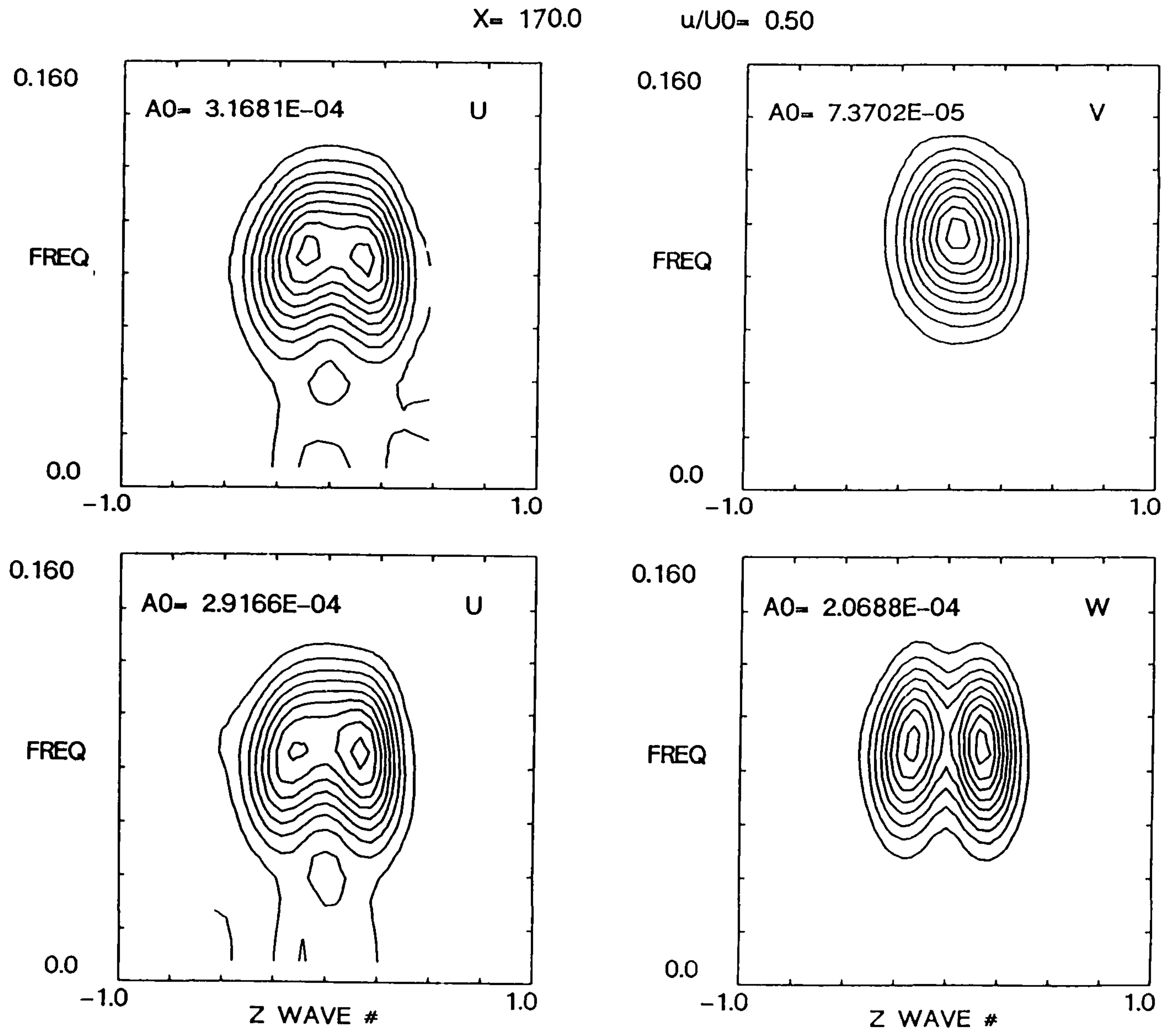
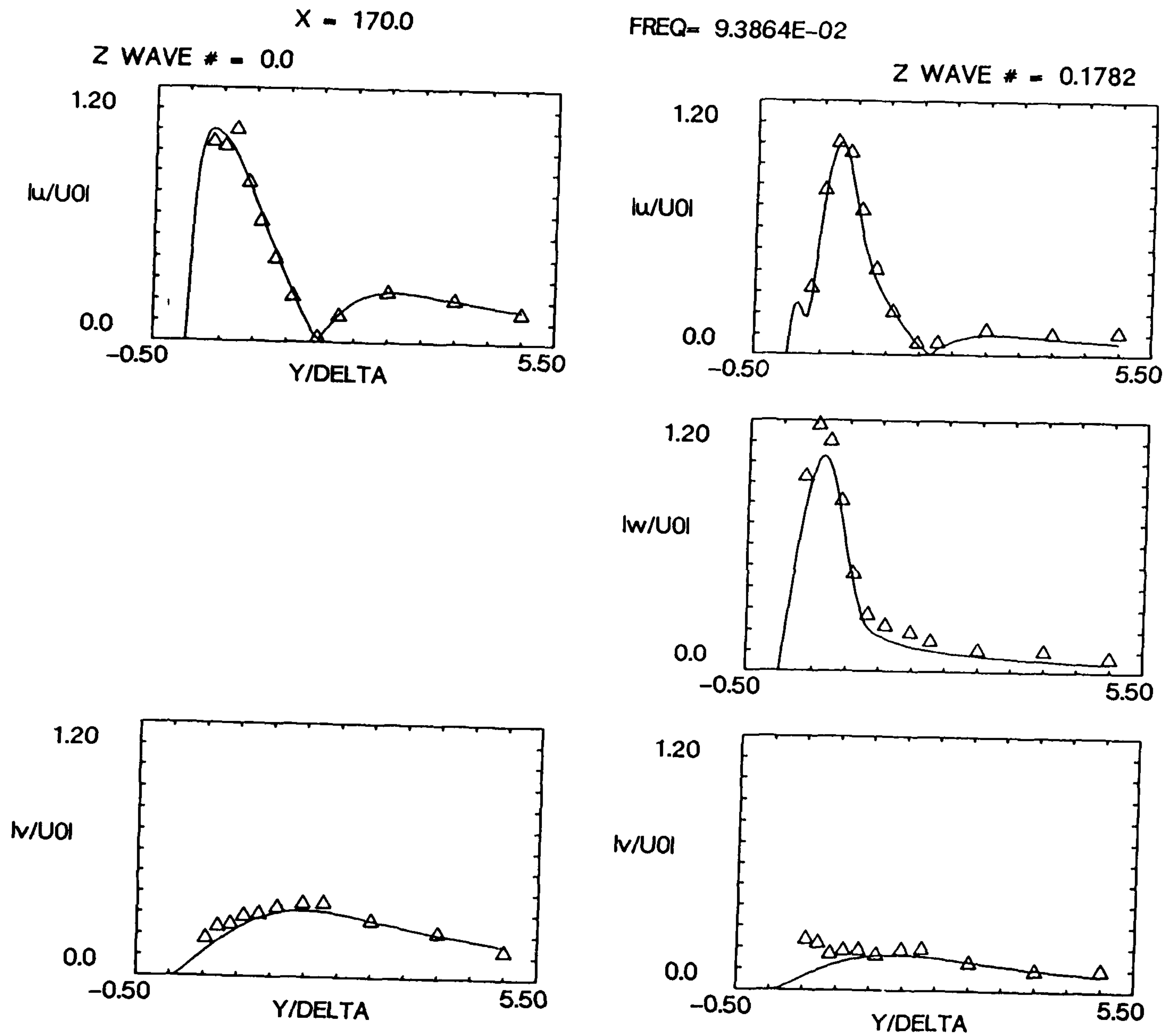

U SPECTRA
W SPECTRA
XD= 280.0
A0= 4.0806E-03
XD= 240.0
A0= 4.6162E-04
XD= 180.0
A0= 3.1870E-04
A0= 6.7465E-04
A0= 4.7572E-04
A0= 3.1854E-04
0.160
FREQ
0.0
-1.0
1.0
Z WAVE #

FIGURE 2



X= 170.0
u/U0= 0.50
0.160
FREQ
0.0
-1.0
1.0
A0= 3.1681E-04
U
A0= 7.3702E-05
V
A0= 2.9166E-04
U
A0= 2.0688E-04
W
Z WAVE #

FIGURE 3



X = 170.0
FREQ= 9.3864E-02
Z WAVE # = 0.0
Z WAVE # = 0.1782
1.20
0.0
-0.50
5.50
|u/U0|
|w/U0|
|v/U0|
Y/DELTA

FIGURE 4a

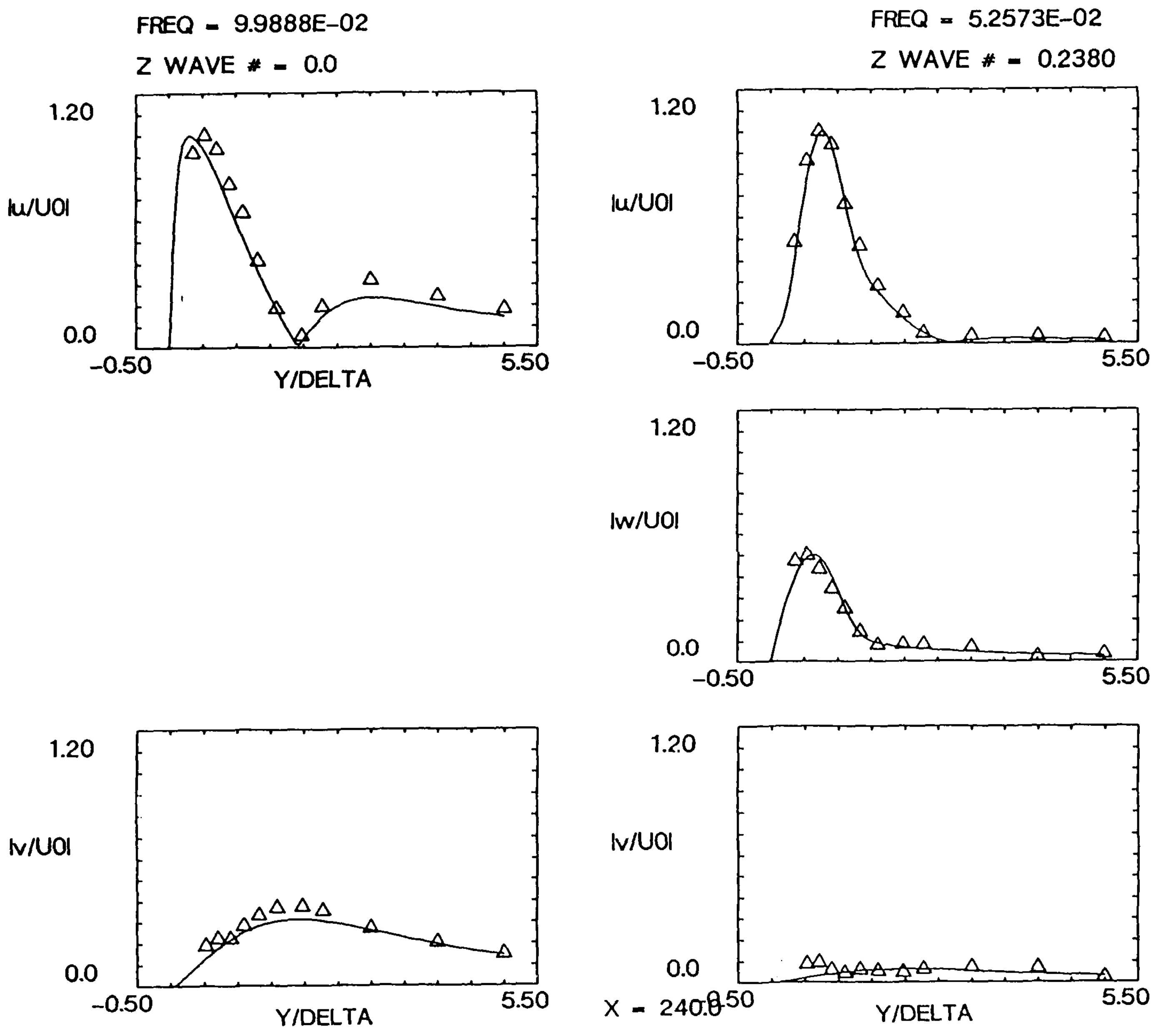

FREQ = 9.9888E-02
Z WAVE # = 0.0
FREQ = 5.2573E-02
Z WAVE # = 0.2380
1.20
0.0
-0.50
5.50
|u/U0|
|w/U0|
|v/U0|
Y/DELTA
X = 240.0

FIGURE 4b

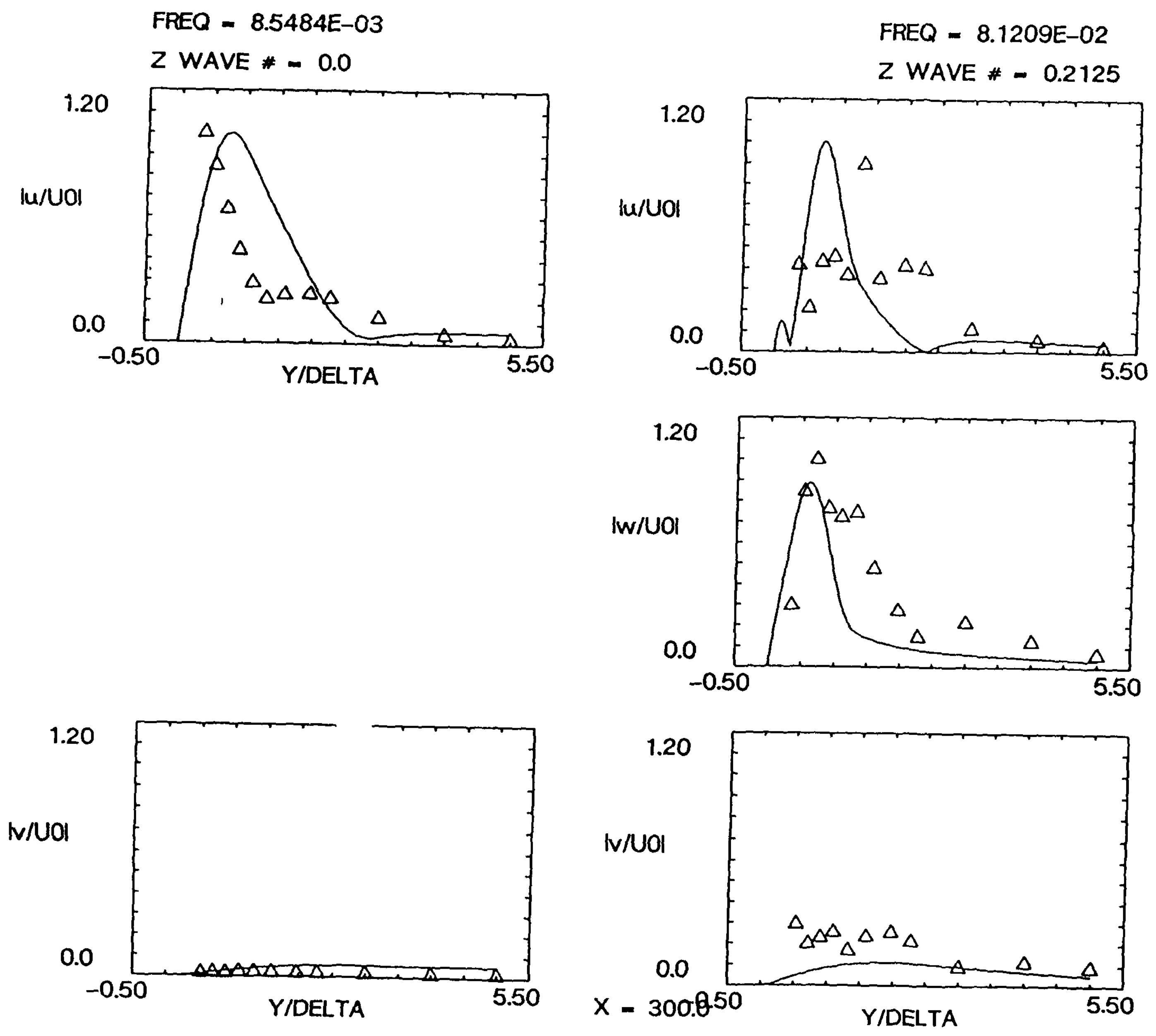

FREQ = 8.5484E-03
Z WAVE # = 0.0
FREQ = 8.1209E-02
Z WAVE # = 0.2125
1.20
0.0
-0.50
5.50
Y/DELTA
lu/U0l
lw/U0l
lv/U0l
X = 300.0

FIGURE 4c

Proceedings, 7th Int. Conf. on Finite Element
Methods on Flow Problems
Univ. of Alabama at Huntsville
Editors: T.J.Chung & G.R.Karr, 1989, pp 17-22

EMBEDDED ADAPTATION FOR TIME DEPENDENT REAL FLUID/GAS FLOWS

Judson R. Baron
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology, Cambridge, MA

ABSTRACT

We review an embedded adaptation algorithm to illustrate applications and requirements for Navier-Stokes and non-equilibrium, unsteady, two-dimensional, flow fields. Both grid refinement and equation modification are included for unstructured distributions of quadrilateral cells. Multiple refinement parameters for distinct physical phenomena serve to monitor the evolving nonuniform field and to recognize flow features in isolated and overlapping subdomains. Modeling time accurate behavior involves tracking of features through a space/time domain; turbulent modeling requires care for an unstructured nonuniform mesh; directional grid refinement conforms to feature geometry. Examples from subsonic, hypersonic, internal and external fields show substantial advantages for the adaptive approach for equivalent numerical accuracies.

1. INTRODUCTION*

A concerted effort has led to the development of adaptive codes that monitor flow fields during computation and simultaneously modify some aspect of the numerical algorithm, thereby achieving a required level of accuracy and resolution without *a priori* knowledge of field details. We specifically review an *embedded* adaptation approach, and extensions to two-dimensional Navier-Stokes, finite rate chemistry, and time accurate descriptions, including their special Euler, perfect gas, and steady state forms.

The motivation is that space and time scales differ appreciably within a given domain. Typical aeronautical applications may involve shock and contact surface discontinuities, stiff relaxation regions due to finite rate chemistry, and thin shear layers adjacent to or separated from surfaces, as flow features which dominate field behavior and are demanding in terms of a numerical method's accuracy and robustness.

The basic concept is that beneficial and/or necessary changes to the physical/mathematical description, the mesh, and the integration algorithm are apparent from the evolving solution field. It is assumed that reasonable initial choices can be made, and that the algorithm both recognizes physics associated with individual scales and provides a basis for the transition from a grid suggested by the configuration and domain to one fitting the fluid events. Changes involve the precision of physical descriptions, mathematical classification, and mesh scale. Most frequently

*Acknowledgement: I am indebted to my students Drs. J.F. Dannenhoffer, J.G. Kallinderis, M.M. Pervais, and Lt. M. J. Aftosmis, for the development of adaptation schemes and solutions. Sponsors were the Air Force Office of Scientific Research, NASA Langley Research Center, and AFWAL Flight Dynamics Laboratory.

adaptation has implied grid refinement so as to reduce the truncation error or to better resolve a feature after identifying the locale and its essential shape.

Clearly adaptation procedures introduce their own scales. The local insertion of fine cells results in mesh scale discontinuities both spatially and temporally, i.e. interfaces in a subdivided domain. Similarly, the effect of embedding is to reinitialize the field; this implies a non-unique strategy and need for special care for accurate transient evaluations. Such interactions between scheme, procedure and influence has led to several suggested zonal, overlay, mixed topology, and hybrid (mesh enrichment and grid displacement) approaches, e.g. [1-7], in part overcoming distortions that arise for simple mesh redistributions.

2. BASIC INTEGRATOR, DETECTOR, AND ADAPTOR METHODS

Ideally the integration scheme should be independent of both the specific adaptation algorithm and the mesh structure, the former simply to be portable, the latter to ensure grid refinement due solely to physics. The basis here is a Lax-Wendroff, finite difference, finite volume, multiple grid accelerator scheme [8,9] that has been modified from its original perfect gas 2-D Euler form to a Navier-Stokes version [10,11], and separately with nonequilibrium conservation relations added [13-14].

The nonuniform field provides a measure of the importance of specific physical phenomena, their recognition, and reasonable parameter variations on which decisions may be based. For example, departures of density differences from an average over all cells prove quite useful in adjusting grid scales for inviscid compressible flow and discontinuities; divided velocity differences are appropriate for shear layers; multiple refinement parameters are necessary when different physical features are present. Providing a variable grid scale achieves a smooth solution overall [12]. In all cases the *embedding* adds grids and nodes to a fixed topology, and essentially aligns the refined region to the feature contour without reducing resolution or accuracy *elsewhere*. In doing so the embedding is characteristically either *external* or *internal* to each feature in accord with the inequality ratio of feature and cell scales, $\ell_F/\ell_C < or > 1$.

A number of techniques and strategies result from the non-unique coupling of physics, background topology, initial coarseness, and refinement. *External* adaptation essentially captures a feature and adjusts to the surrounding field according to it own scale. An *internal* adaptation may dominate the demands on data storage. In either case, directional rather than isotropic subdivision of cells provides appreciable savings. Often preembedding is clearly advantageous, such as when surface shear regions or moving discontinuities are evident. Alternatively, a nested sequence of external adaptation levels may be imposed simultaneously when a feature is first recognized. Temporal adaptation can be related to Courant number/spatial adaptation constraints, but also may be governed by nonequilibrium time scales. The basis for adaptive sufficiency may be continued refinement until convergence results for a global parameter, e.g. lift, rather than local residuals, consistent with the concept of capturing interacting features in sufficient detail.

3. EXAMPLES FROM VISCOUS, NONEQUILIBRIUM, AND HYPERSONIC FLOWS

Several applications illustrate the behavior of the algorithm when focused on different physical features. Fig. 1 shows a basic inviscid example of four adaptation cycles carried out for a NACA-0012 airfoil section at $M_\infty = 0.85$ and $1.0°$ angle of attack [9]. Adaptive cycles were imposed on convergence to specific residual levels, here $2. \times 10^{-4}$, and monitoring of global lift and drag aerodynamic coefficients resulted in 4 cycles.

Reacting inviscid flow requires active refinement parameters for nonequilibrium variables as well as density and provides another example of external embedding. Very stiff situations can be associated with the unsteady case, such that resolution demands are crucial even with implicit modeling of chemical source terms. Temporal adaptation takes on special importance, and has been implemented for a shock tube (Fig. 2) to illustrate how the space/time adaptation involves embedding and later removal after passage of the several disturbance regions [13].

A generalized schematic of the uncoupled space/time grid refinement is also shown for two space dimensions, with time steps limited by either convective or reactive constraints, and is the background for the momentary grid that corresponds to a shock interacting with a circular arc configuration (Fig. 2) in a dissociating oxygen medium [14]. The grid importance, and the equivalence of an adapted and globally fine mesh, is evident from Fig. 3.

Viscous features require *internal* embedding and usually have been preembedded (clustered) when shear layer locations and their extents are known. The procedural concern is with unstructured cell independence for discreteness evaluations of stress contributions, and with limitations on structured concepts that are inherent in turbulence modeling. Fig. 4 illustrates adaptation applied to a RAE 2822 airfoil transonic field which includes attached and separated turbulent shear layers as well as shocks. An assumed Baldwin-Lomax model was interpreted in cell centered fashion with adjustments and interpolations to meet requirements along surface normals [12]. The grid in Fig. 4 shows the characteristic dense concentration along shock, surface and wake regions, the latter being in accord with the minimum velocity locus. On the enlarged trailing edge portion in the lower part of the figure the superimposed crosses indicate embedded interface locations, and the presence of smooth Mach number contours in the multiply adapted confined region is satisfying.

Another example involving multiple features is that of a hypersonic shock layer (Fig. 5). Inviscid, reacting flow is confined to a thin domain exhibiting both entropy and relaxation layers that are not always separated in space. The free shock boundary implies adaptation being applied within a changing domain during iteration to a steady state. Results for a Mach number 12. example show the coarse grid refinement into a finer mesh downstream of the shock as a result of chemical relaxation, and elsewhere due to rapid expansions and entropy variations.

4. REFERENCES

1. Dwyer,H.A., Kee,R.J. and Sanders,B.R., *Adaptive Grid Method for Problems in Fluid Mechanics and Heat Transfer*, J. AIAA, *18*, 1205-1212 (1980).
2. Rai,M.M. and Anderson,D.A., *Application of Adaptive Grids to Fluid Flow Problems with Asymptotic Solutions*, AIAA Paper 81-0114 (1981).
3. Berger,M.J. and Oliger,J., *Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations*, NYU Courant Institute Manuscript NA-83-02 (1983).
4. Erlebacher,G. and Eiseman,P.R., *Adaptive Triangular Mesh Generation*, AIAA Paper 84-1607 (1984).
5. Arney,D.C. and Flaherty,J.E., *A Two-Dimensional Mesh Moving Technique for Time Dependent Partial Differential Equations*, J. Computational Physics, *67*, 124-144 (1986).
6. Steger,J.L., Dougherty,F.C. and Benek,J.A., *A Chimera Grid Scheme*, Advances in Grid Generation, Ghia, Editor, ASME FED *5* (1983).
7. Lohner, R., *Adaptive Remeshing for Transient Problems with Moving Bodies*, AIAA Paper 88-3737 (1988).
8. Ni,R.H., *A Multiple Grid Scheme for Solving the Euler Equations* , J. AIAA, *20*, 1565-1571 (1982).
9. Dannenhoffer,J.D., *Grid Adaptation for Complex Two-Dimensional Transonic Flows*, M.I.T. PhD Thesis (1987).
10. Kallinderis,J.G. and Baron,J.R., *Adaptation Methods for a New Navier-Stokes Algorithm*, AIAA Paper 87-1167 (1987).
11. Kallinderis,J.G. and Baron,J.R., *Unsteady and Turbulent Flow using Adaptation Methods*, Proceedings 11th Intl. Conf. on Numerical Methods in Fluid Dynamics, Williamsburg (1988).
12. Kallinderis,J.G. and Baron,J.R., *Adaptation Methods for Viscous Flows*, Computational Methods in Viscous Aerodynamics, C.A. Brebbia, Editor (1988).
13. Pervais,M.M. and Baron,J.R., *Temporal and Spatial Adaptive Algorithm for Reacting Flows*, Communications in Applied Numerical Methods, *4*, 97-111 (1988).
14. Pervais,M.M., *Spatio-Temporal Adaptive Algorithm for Reacting Flows*, M.I.T. PhD Thesis (1988).
15. Aftosmis,M.J. and Baron,J.R., *Adaptive Grid Embedding in Nonequilibrium Hypersonic Flows*, AIAA 24th Thermophysics Conference, Buffalo (1989).

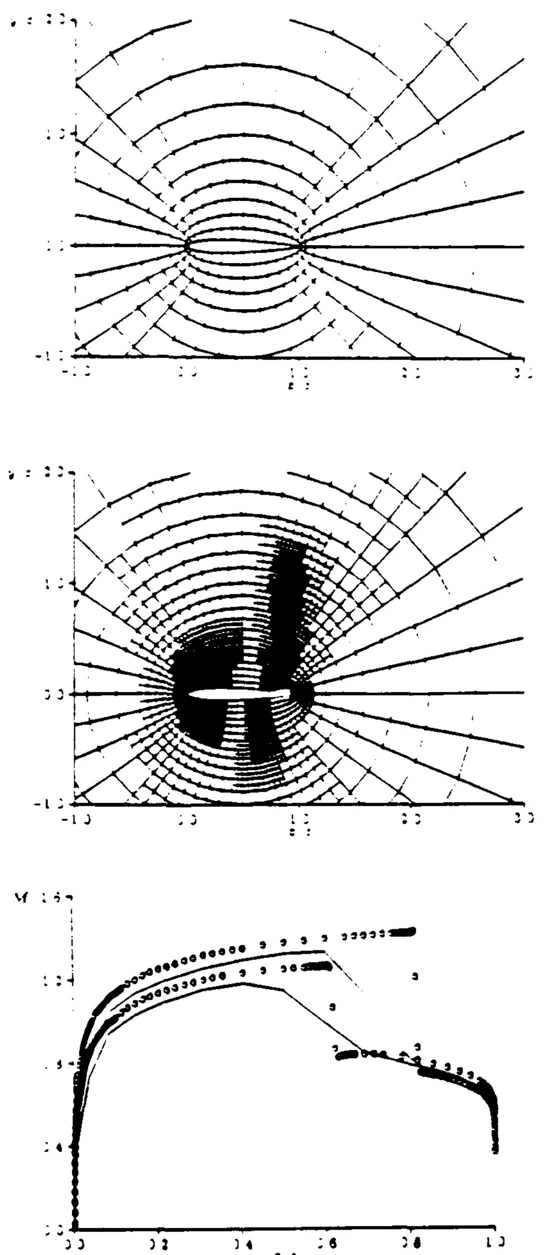


Fig. 1: NACA-0012 Airfoil, Inviscid Flow at $M_\infty = 0.85$ and $\alpha = 1\,\text{deg}$. *Top*: Initial Coarse Grid *Middle*: Final, Four Adaptations Grid *Bottom*: Surface Mach Number Distributions (Line,Symbols) are Solutions on (Initial, Final) Grids.

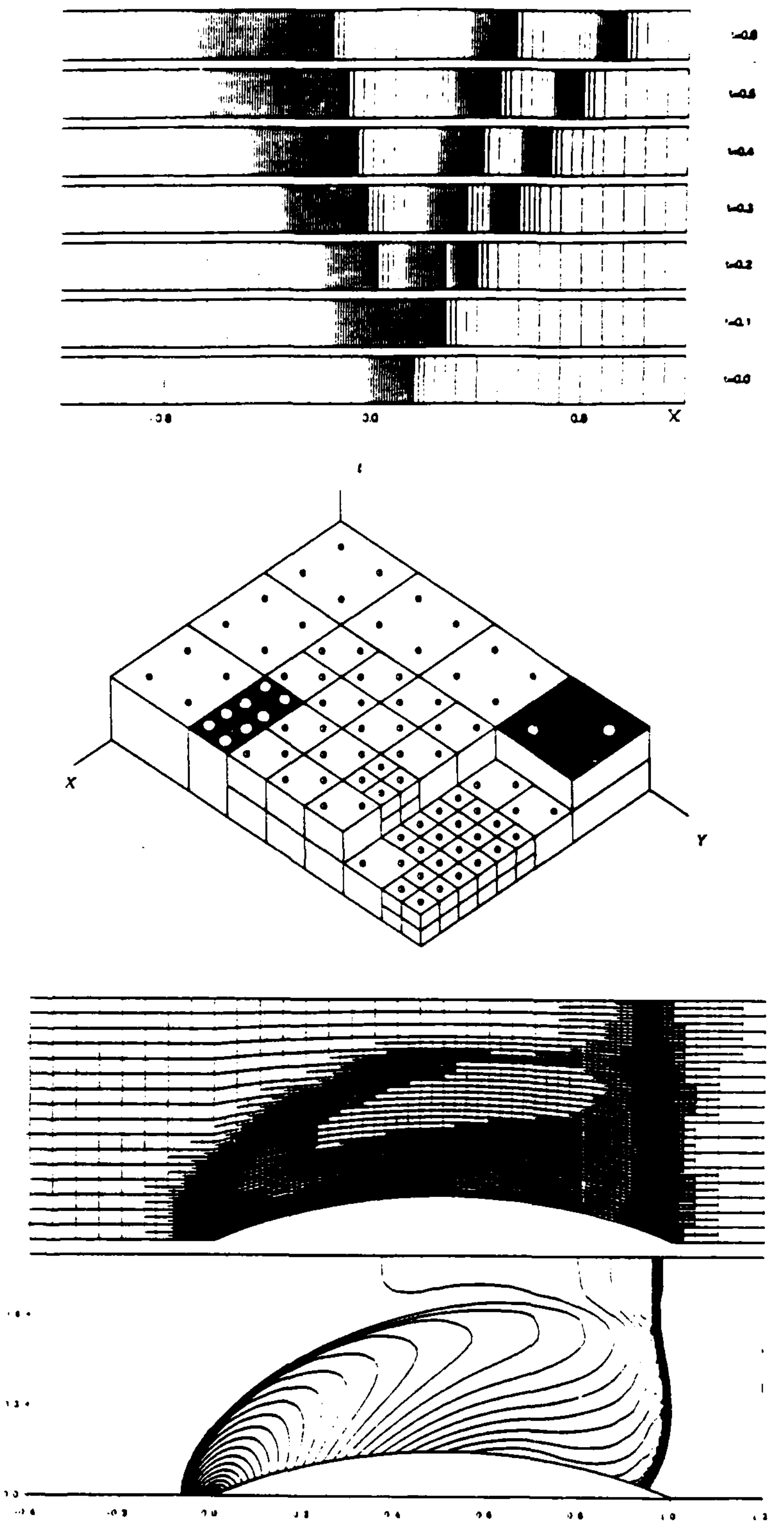


Fig. 2 Temporal Adaptation. *Top*: Shock Tube Spatial Grid Variation with Time. *Middle*: Schematic of Three Dimensional Adapted Cell Distribution in Space/Time. *Bottom*: Reacting (Dissociating Oxygen) Flow for $M = 2$. Transient Shock over 15% Circular Arc Section. Grid and Density Contours, $t = 0.6$.

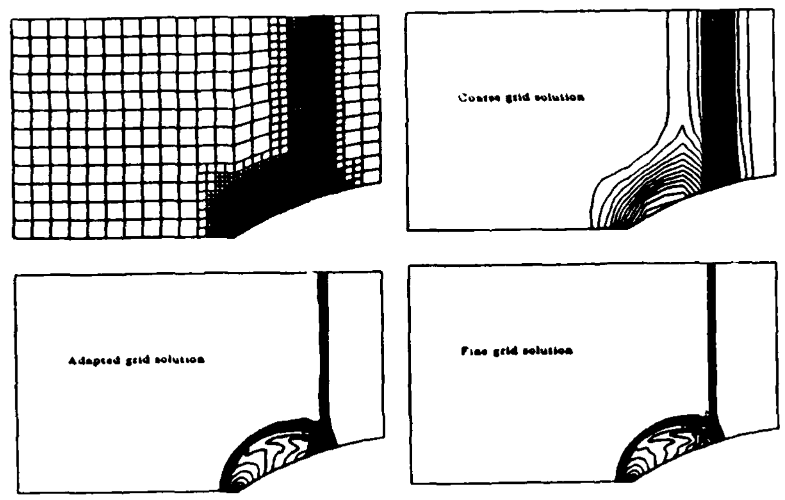


Fig. 3 Density Contours for Transient ($t = 0.3$) Shock Flow on Coarse, Fine and Adapted Grids.

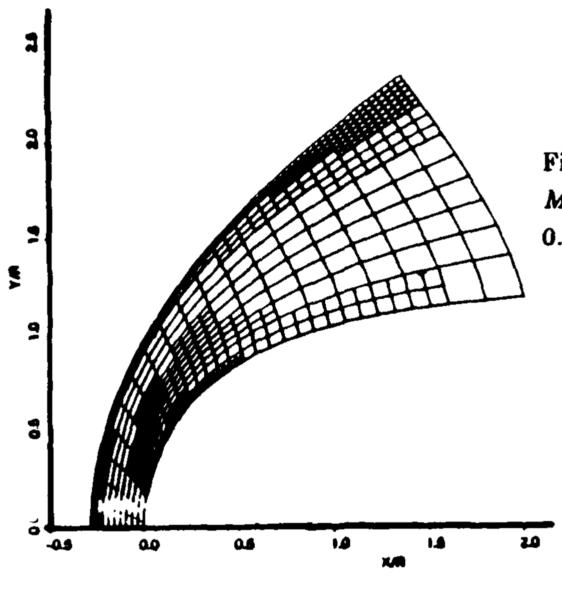


Fig. 5 Adapted Hypersonic Shock Layer, $M_\infty = 12.$, Dissociating Oxygen, $R_N = 0.01m$, 40 km.

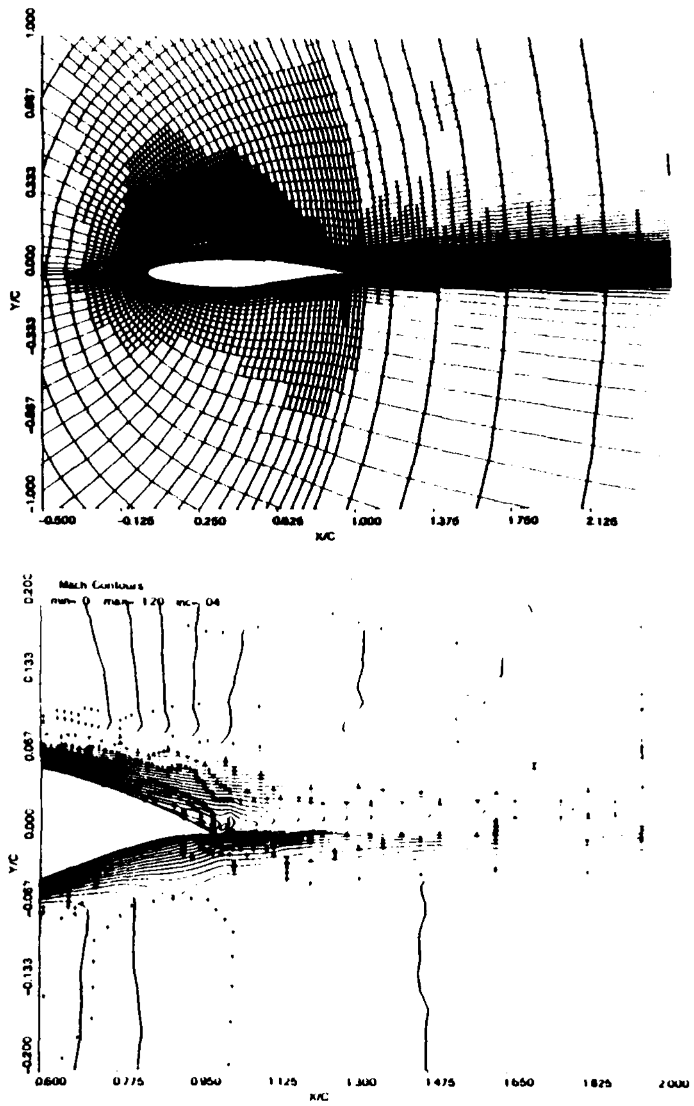


Fig. 4 RAE 2822 Airfoil, Turbulent Boundary Layer, $M_\infty = 0.725$, $Re = 6.5 \times 10^6$, $\alpha = 2.44$ deg. *Top*: Grid for 3 Adaptation Levels, *Bottom*: Mach Contours and Interface Markers for Separated Flow Near Trailing Edge.

AIAA'89

APPENDIX J I

AIAA-89-0655

Higher-Order and 3-D Finite Element Methods for the Euler Equations

Richard A. Shapiro

United Technologies Corporation
c/o Thinking Machines, Inc.
245 First St.
Cambridge MA 02142

Earll M. Murman

Computational Fluid Dynamics Laboratory,
Department of Aeronautics and Astronautics,
Massachusetts Institute of Technology
Cambridge MA 02139

Higher-Order and 3-D Finite Element Methods for the Euler Equations

Richard A. Shapiro*

United Technologies Corporation

Earll M. Murman†

Computational Fluid Dynamics Laboratory,

Department of Aeronautics and Astronautics,

Massachusetts Institute of Technology

1 Abstract

A finite element algorithm for solving the steady Euler equations is presented. The extension of this algorithm to biquadratic elements in two dimensions is discussed, and results verifying the method are presented. Examples which demonstrate the computational savings realisable with biquadratic elements are shown. A method of degenerating a biquadratic element into 4 bilinear elements to improve shock capturing properties is developed. The extension of the algorithm to 3-D is discussed. Examples of flow in a corner and flow in a scramjet inlet are shown.

2 Introduction

Numerical solution of the Euler equations describing the dynamics of an inviscid, compressible, ideal gas are becoming an important tool for the practicing aerodynamicist [1]. In recent years, many methods suitable for unstructured grid have been introduced, including the Galerkin finite element algorithm [2,3,4] and the cell-vertex finite element algorithm [5,6,7]. The main advantage of the unstructured grid methods is geometric flexibility. They allow complex geometries to be treated in a straightforward manner, and allow one to use grid embedding with relative ease.

This paper explores two finite element ideas new to the solution of the steady Euler equations. The first idea is the use of higher-order (biquadratic) finite elements in two dimensions, and the second idea is the use of adaptive, hexahedral finite elements in three dimensions.

*AIAA Member

†AIAA Fellow

A mesh of biquadratic elements requires far fewer elements for a given accuracy than the a mesh of bilinear elements. Biquadratic finite elements have been applied to the Euler equations with limited success previously [8], but we present a formulation which produces excellent results and significant computational savings over bilinear elements (with or without adaptation) for many problems. Preliminary results were reported in [4]. These and later examples demonstrate that the biquadratic elements can exhibit some mild oscillation at discontinuities. To try and correct the oscillation problem, the use of a mixture of biquadratic and bilinear elements in the same problem by adaptively degenerating a single biquadratic element into four bilinear elements in regions near discontinuities was explored. Test problems include channel flow over a circular arc bump, flow in a converging channel (including comparisons with the exact solution), and flow in a scramjet inlet.

In three dimensions, adaptation with grid embedding using hexahedral finite elements is demonstrated. Previous work on adaptive methods for the Euler equations using grid embedding in two dimensions includes the work of Dannenhoffer [9], Löhner [10], Shapiro and Murman [4], and Oden [11]. In three dimensions, the primary adaptive methods have been the grid movement methods [12], and the grid regeneration methods [13,14]. To our knowledge, this work is the first use of adaptation by grid embedding for hexahedral elements. Test examples showing flow in a corner and flow in a 3-D scramjet inlet will be presented.

3 Solution Algorithm

This section briefly describes the solution algorithm used for the 3-D trilinear and 2-D biquadratic elements. For a more complete discussion of the algorithm, see [15]. The Euler equations describing the flow of an inviscid, compressible fluid in two dimensions can be written in conservative form as

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho e\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ \rho u h_0\end{bmatrix}+\frac{\partial}{\partial y}\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ \rho v h_0\end{bmatrix}=0, \quad (1)$$

where where e is total energy, p is pressure, ρ is density, u and v are the flow velocities in the x and y directions, and h_0 is the total enthalpy, given by the thermodynamic relation

$$h_0 = e + \frac{p}{\rho}. \qquad (2)$$

In addition, one requires an equation of state in order to complete the set of equations. For an ideal gas, this can be written

$$\frac{p}{\rho} = (\gamma - 1)\left[e - \frac{1}{2}(u^2+v^2)\right], \qquad (3)$$

where the specific heat ratio ($\gamma = 1.4$) is constant for all calculations reported.

It is convenient to write the equations in vector form as

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = 0, \qquad (4)$$

where $\mathbf{U}$ is the vector of state variables, and $\mathbf{F}$ and $\mathbf{G}$ are flux vectors in the x and y directions, corresponding to the vectors in Eq. (1) above. To expand these equations to 3-D, a z-momentum equation and z flux term must be added to the above equations.

3.1 Spatial discretization

The finite element approach to discretising these equations divides the domain of interest into elements determined by some number of nodes. In general, these elements can be of any shape. In this paper, we have chosen to use quadrilateral elements in two dimensions and hexahedral elements in three dimensions. The algorithm permits the degeneration of quadrilaterals into triangles and hexahedra into other shapes, so this is not a major geometric restriction. Quadrilateral and hexahedral elements were chosen since grids of these elements require fewer elements for a given number of nodes than a comparable triangular or tetrahedral grid, resulting in some storage and/or CPU savings. In the current implementation, 9 nodes are used for biquadratic elements and 8 nodes are used for 3-D elements. The 9-node element is a *subparametric* element, with the geometry interpolated bilinearly and the fluxes and state vectors interpolated biquadratically. Figure 1 shows the geometry of the 9-node element, and shows how the physical coordinates are mapped to natural coordinates at the element level. The spatial discretisation method begins with the Euler equations in conservative form (Eq. (4)). Within each element the state vector $\mathbf{U}^{(e)}$ and flux vectors $\mathbf{F}^{(e)}$, $\mathbf{G}^{(e)}$ and $\mathbf{H}^{(e)}$ are written

$$\begin{aligned}\mathbf{U}^{(e)} &= \sum N_i^{(e)}\mathbf{U}_i, &(5)\\ \mathbf{F}^{(e)} &= \sum N_i^{(e)}\mathbf{F}_i, &(6)\\ \mathbf{G}^{(e)} &= \sum N_i^{(e)}\mathbf{G}_i, &(7)\end{aligned}$$

where $\mathbf{U}_i, \mathbf{F}_i$, and $\mathbf{G}_i$ are the nodal values of the state vector and flux vectors, and $N_i^{(e)}$ is the set of interpolation functions for element e.

These expressions can be differentiated to obtain formulas for the derivative of a quantity in each element in terms of the nodal values of that quantity and the geometry of the element. In all the following steps, the two-dimensional algorithm will be shown for simplicity, with the basic steps identical for three dimensions.

The expressions for the derivatives are substituted into equation (4) and summed over all elements to obtain

$$\mathbf{N}_i\frac{d\mathbf{U}_i}{dt} = -\frac{\partial \mathbf{N}_i}{\partial x}\mathbf{F}_i - \frac{\partial \mathbf{N}_i}{\partial y}\mathbf{G}_i \qquad (8)$$

where $\mathbf{N}_i$ is now a global *row* vector of interpolation functions, determined by summing the interpolation functions for each element.

It is impossible to make Eq. (8) hold for all points in space (since the space of interpolation functions does not include all solutions to the Euler equations), so some "average" solution is required. The next step creates a *weak* form of the equations. This can be thought of as a projection onto the space spanned by some other row vector of functions $\tilde{\mathbf{N}}$, called *test functions*, such that the error in the discretisation is orthogonal to the space spanned by the test functions. In the weak form, the equation is no longer required to be satisfied pointwise, but instead the equation is required to hold for each test function. This allows the introduction of discontinuous solutions, as well as providing some means for obtaining the nodal values of the unknowns. To create this weak form, premultiply Eq. (8) by $\tilde{\mathbf{N}}^T$ and integrate over the entire domain. When this is done, one obtains

$$M\frac{d\mathbf{U}_i}{dt} = -\iint(\tilde{\mathbf{N}}^T\frac{\partial \mathbf{N}}{\partial x}\mathbf{F}_i + \tilde{\mathbf{N}}^T\frac{\partial \mathbf{N}}{\partial y}\mathbf{G}_i)\,dV, \qquad (9)$$

$$M = \iint \tilde{\mathbf{N}}^T\mathbf{N}\,dV, \qquad (10)$$

which results in the semi-discrete equation

$$M\frac{d\mathbf{U}_i}{dt} = -R_x\mathbf{F}_i - R_y\mathbf{G}_i, \qquad (11)$$

where M is the consistent mass matrix, and R_x and R_y are residual matrices. The matrices M, R_x and R_y involve the integration of quantities over the domain. These integrations are done at the element level in natural (element-based) coordinates, and assembled to give the global matrices.

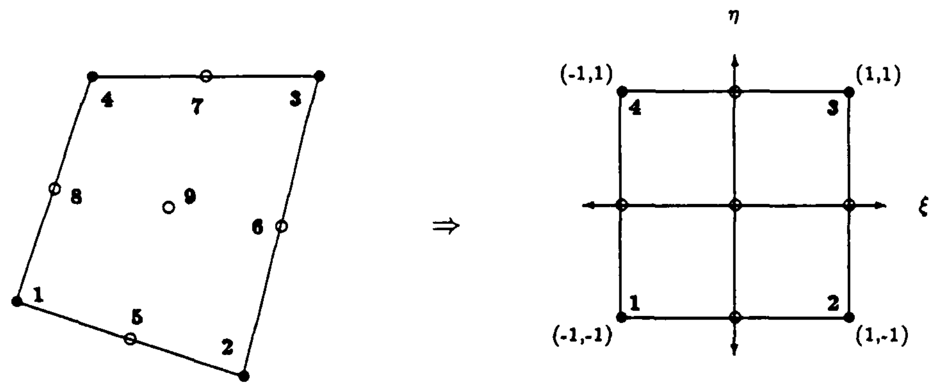


Figure 1: Geometry of Two-Dimensional Element

As derived, Eq. (11) gives a coupled set of ODE's to solve for the nodal values of the state vector. The mass matrix M is sparse and positive definite but unstructured, so that its inversion requires considerable computational effort. If one is only interested in the steady state, M can be replaced by a "lumped" (diagonal) matrix M_L, where each diagonal entry is the sum of all the elements in the corresponding row of M. This allows Eq. 11 to be solved explicitly. If one wishes to solve the unsteady Euler equations, it is better to invert the mass matrix with a few iterations of a preconditioned conjugate-gradient solver [16].

3.2 Solid Surface Boundary Condition

At walls, the portions of the flux vectors representing convection normal to the wall are set to zero before each iteration, and flow tangency is enforced after each iteration. The equation for the fluxes is then

$$\mathbf{F}_w = \begin{bmatrix} \rho u_m \\ \rho u u_m + p \\ \rho u_m v \\ \rho u_m h \end{bmatrix} \qquad \mathbf{G}_w = \begin{bmatrix} \rho v_m \\ \rho u v_m \\ \rho v v_m + p \\ \rho v_m h \end{bmatrix} \tag{12}$$

where u_m and v_m are corrected velocities such that the total convective contribution normal to the wall is 0. These velocities are the x and y components of the tangential velocity, given by

$$u_m = u(1-n_x^2) - v n_x n_y, \tag{13}$$
$$v_m = v(1-n_y^2) - u n_x n_y, \tag{14}$$

where n_x and n_y are the components of the unit normal at the node. This is easily derived from the vector expression

$$\vec{v}_{tan} = \vec{v} - (\vec{v}\cdot\hat{n})\hat{n}, \tag{15}$$

where $\hat{n}$ is the unit normal to the wall. This expression is obtained by finding the normal component of the velocity $(\vec{v}\cdot\hat{n})$, and subtracting it from the velocity vector.

3.3 Far-Field Boundary Condition

A one-dimensional characteristic treatment is used on the open or far-field boundary. From the inward-directed unit normal vector $\hat{n}$, the unit tangent vector $\hat{t}$ and the normal and tangential velocities u_n and u_t are calculated. The 1-D Riemann invariants (and the corresponding wave speeds) are

$$\text{inv.:} \begin{bmatrix} \frac{2a}{\gamma-1} + u_n \\ \frac{2a}{\gamma-1} - u_n \\ \frac{p}{\rho^\gamma} \\ u_t \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix}, \text{speeds:} \begin{bmatrix} u_n + a \\ u_n - a \\ u_n \\ u_n \end{bmatrix}. \tag{16}$$

At each point on the boundary, the invariants are calculated using the solution state vector $\mathbf{U}$ and the "free stream" state vector $\mathbf{U}_\infty$. Then, a decision is made based on the sign of the corresponding wave speed (from the interior u_n) whether to use the invariant based on the current state, or the invariant based on the free stream. If the relevant wave speed is positive (at a supersonic inflow boundary, for example, all 4 characteristic speeds are positive), then the free stream value is used for that characteristic.

The invariants are transformed back into a set of primitive variables, and these primitive variables are used to calculate the fluxes at the boundary nodes for use in the residual calculation. After the complete iteration, the characteristics are also used to update the state vectors at the boundary. Although this is not necessary for convergence, updating the state vectors after each iteration improves the robustness of the algorithm, especially for biquadratic elements.

For many problems with subsonic outflow, particularly problems involving choked flow, evaluating C_1 based on the free stream is not an appropriate boundary condition. Specifying exit pressure is a more physical condition, and this conditions is enforced by setting C_1 such that when the characteristics are transformed back into primitive variables, the desired pressure is obtained.

3.4 Smoothing

To capture shocks and stabilise the scheme, an artificial viscosity needs to be added. Currently, the smoothing used consists of a fourth-difference term and a pressure-switched second-difference term. Due to the unstructured nature of the grids, a Laplacian-type of second-difference is used, instead of normal and tangential or ξ and η differences.

The heart of the smoothing method is the calculation of an elemental contribution to a second difference. For the three-dimensional elements and the 2-D sub-elem 's described below, the second difference of a quantity q is calculated by the following procedure:

1. In each element, calculate an elemental average of q by summing all the nodal values of q and dividing by the number of nodes in the element.
2. The contribution of the element to the second difference at node i is $W(q_{\text{avg}} - q_i)$, where W is some elemental weighting factor (often 1).
3. Sum the elemental contributions to a node over all elements containing that node.

For the biquadratic element, the main element is divided into 4 sub-elements (Fig. 2) and a second difference is calculated on the sub-elements as described above. To calculate the complete smoothing for a time step, one first calculates the nodal second difference of pressure. This is turned into an elemental quantity by simple averaging over all the nodes in an element. The elemental second difference is then normalised by an elemental pressure average to form an elemental weight for the following step. The second-difference smoothing term is the weighted second difference of the state vectors, multiplied by a constant between 0 and 0.05. The fourth-difference smoothing term is the second difference of the second difference of the state vectors multiplied by a constant between 0.001 and 0.05. The sum of

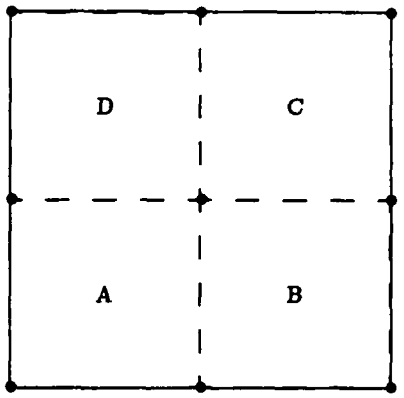


Figure 2: Division of biquadratic element into 4 subelements (A,B,C and D)

these two terms is added directly into the time integration of Eq. 17.

3.5 Time Integration

To integrate equation (11), the following multi-step method is used:

$$\begin{aligned}
U_i^{(1)} &= U_i^n + \frac{1}{4}\lambda(-\frac{\Delta t_i}{M_{L_i}} R_i(U^n) + V_i^n) \\
U_i^{(2)} &= U_i^n + \frac{1}{3}\lambda(-\frac{\Delta t_i}{M_{L_i}} R_i(U^{(1)}) + V_i^n) \\
U_i^{(3)} &= U_i^n + \frac{1}{2}\lambda(-\frac{\Delta t_i}{M_{L_i}} R_i(U^{(2)}) + V_i^n) \qquad (17) \\
U_i^{(4)} &= U_i^n + \lambda(-\frac{\Delta t_i}{M_{L_i}} R_i(U^{(3)}) + V_i^n) \\
U_i^{n+1} &= U_i^{(4)}
\end{aligned}$$

where $R_i(U)$ is the right-hand side of Eq. (11) with the fluxes based on state vector U, V_i is the smoothing term described above, M_{L_i} is the entry in the lumped mass matrix for node i, and λ is the CFL number. Local time stepping is used to accelerate convergence, with the time step given by

$$\Delta t_i = \frac{\Delta x_i}{|u| + a} \qquad (18)$$

where Δx_i is some nodal characteristic length, and u is the flow velocity at the node. In this algorithm, for Δx_i we use the minimum (over all elements containing the node) of the average lengths of opposite sides of the element.

3.6 Choice of Test Functions

Various choices for $\tilde{N}$ are possible, each giving rise to a particular spatial discretisation. The details of the selection, and the advantages and disadvantages of each, are discussed more fully in [15]. Briefly, if one choses $\tilde{N}^{(e)} = N^{(e)}$ one obtains the Galerkin finite element approximation. This is the "traditional" finite element method, and is the method used for the biquadratic elements. Some of the 2-D, linear examples shown in this paper also make use of the Galerkin method.

One can also choose $\tilde{N}^{(e)}$ constant, which results in the cell-vertex approximation. This is the approximation used in three dimensions, and for some of the linear 2-D examples shown. It was not used for the biquadratic elements, as it appears to be unstable. The likely cause of this instability is that the mid-element node in the biquadratic element is decoupled from the other 8 nodes in the element. That is, the flux at the center node does not enter into the computation of the residuals at the other 8 nodes.

For bilinear elements, the Galerkin and cell-vertex methods produce almost identical answers for a wide range of problems [4,15], and so no further distinction between these methods will be made in this paper for the 2-D, bilinear solutions presented.

4 Comparison of Bilinear and Biquadratic Elements

Biquadratic elements have the potential to produce more accurate solutions on coarser meshes. The use of coarser meshes may allow the CPU cost for a given accuracy to be reduced. A single biquadratic Galerkin element requires about a factor of 3.3 more computation than a single bilinear Galerkin element. Since one can view a biquadratic element as replacing 4 bilinear elements (for a given number of nodes), there is a slight savings attained here. In many cases, however, a single biquadratic element can produce results comparable to 16 bilinear elements, resulting in substantial computational savings. Unfortunately, the biquadratic elements have some slight problems near shocks due to Gibb's phenomenon. Three test problems are used to compare the bilinear and biquadratic formulations of the Galerkin method in two dimensions. These problems are 5° converging channel flow at Mach 2, the flow over a 10% cosine-squared bump in a channel at Mach 0.5, and the flow in a scramjet inlet at Mach 3. The first and third cases demonstrate the ability of the biquadratic elements to handle flows with shocks. The second case demonstrates the ability to capture smooth flows.

4.1 5° Channel Flow

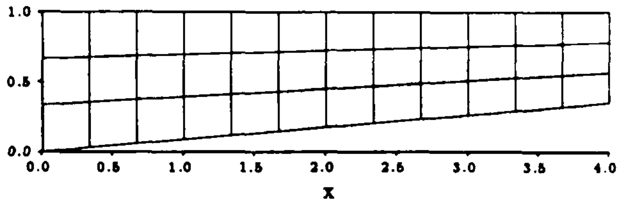


Figure 3: 5° Converging Channel, 12x3 grid

Case	RMS % Error in ρ	CPU (seconds)
40x10 Bilinear	1.9	19
80x20 Bilinear	1.1	126
12x3 Biquadratic	2.1	9
20x5 Biquadratic	1.3	26
40x10 Biquadratic	0.7	150

Table 1: Comparison of Bilinear and Biquadratic Galerkin Solutions to 5° Wedge Problem

The 5° channel flow was computed using biquadratic elements on 12x3 and 40x10 grids. The 12x3 grid is shown in Figure 3. The densities along the surface for these cases, along with the exact solution (dotted line), are shown in Figs. 7 and 8. Note the sharpness of the shocks and the excellent agreement with the exact solution. For comparison purposes, the solution using the bilinear Galerkin method is shown for a 40x10 grid in Fig. 9 and for an 80x20 grid in Fig. 10. Note the smearing and low-wavenumber oscillations present near the shocks. These are due mainly to dispersive error, and are not the result of artificial viscosity [15].

Table 1 presents a quantitative comparison of average error and computational effort for these cases. This table shows some interesting facts. For about half the effort required in 40x10 bilinear case, one can obtain the same accuracy using a 12x3 biquadratic mesh. For about the same effort as an 80x20 bilinear mesh, a 40x10 biquadratic mesh provides better accuracy.

4.2 Subsonic, Smooth Flow

One expects the biquadratic elements to be very good for smooth flows. To verify this, $M_\infty = 0.5$ flow over a 10% cosine-squared bump was computed on a 24x8 biquadratic mesh and a 60x20 bilinear mesh. Figure 11 shows contours of density for the biquadratic elements. The contours are quite symmetric, as one would expect from a flow which remains completely subsonic. Most of the non-smoothness

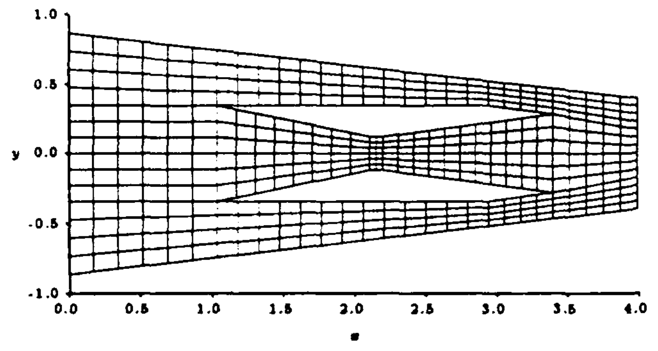


Figure 4: Two-Dimensional Scramjet Inlet Initial Grid, 416 Elements

seen in the contours is introduced by the plot package (which divides each biquadratic element into 32 linear triangles), rather than actual errors in the flow. For comparison, Fig. 12 shows these contours in the bilinear case. The agreement is quite good, and the biquadratic case required about 3/4 the CPU time of the bilinear case.

4.3 Mach 3 Flow in a Scramjet Inlet

The flow in a scramjet inlet at $M_\infty = 3$ was computed using adaptive bilinear elements and with biquadratic elements. The geometry for this case is roughly that used in Kumar's paper [17], and the initial grid for the bilinear case (and the only grid for the biquadratic case) is shown in Fig. 4. The computed density solution for the bilinear case is shown in Fig. 13. There are several interesting features to observe here. Note that the shock off the inlet entrance reflects six times, and in particular note that near $x = 3.7$, $y = 0.3$, the shock bends slightly toward the center line. Now look at the strut leading edge shocks, and note that the third reflection of the shocks (starting near $x = 2.8$, $y = \pm 0.3$, and barely visible as "dents" in the contour lines) passes through the sixth reflection of the entrance shock where the entrance shock experiences the bend. Another cause of the bending is the interaction with the slip line off the trailing edge. This slip line is not readily apparent in Fig. 13, but a plot of the total pressure loss (not shown) at the exit clearly indicates its presence.

Density contours for the solution with biquadratic elements are shown in Fig. 14. The resolution is not as good as Fig. 13, but the bilinear case used 14648 elements and required 51 minutes to compute on a 3-processor Alliant FX/8, while the biquadratic case used 416 elements and required 2 minutes to compute. This case further verifies the usefulness of the biquadratic elements.

5 Adaptive Degeneration of Biquadratic Elements

The biquadratic elements formulated here perform quite well, even for flows with shocks. In some cases, the mild oscillations seen near shocks may be highly undesirable. To overcome this problem, an approach is taken in which the element residuals (R_x and R_y, see Eq. 11) near a shock are computed as if the biquadratic element were 4 bilinear Galerkin elements. This idea has some similarities to the flux-corrected transport methods [18], but it is much simpler to implement. There are two issues in this procedure: deciding which elements should be degenerated into linear elements, and treating the interface between a bilinear region and a biquadratic region.

The decision to degenerate an element requires the formation of a shock detector. For this study, we tried a very simple shock detector consisting of a scaled second difference of pressure. This selection was motivated by the choice of a switch for the non-linear second-difference smoothing contribution above. To calculate the switch, the absolute value of the nodal second difference of pressure is averaged over each element, and this average is divided by the average pressure in the element. Where this quantity exceeds a certain threshold, the element is degenerated. This indicator gives reasonable results for the very limited class of prob ems tried so far, and more sophisticated indicators are under study.

The degeneration of elements into linear elements introduces interfaces between the bilinear regions and the biquadratic regions. In order to maintain conservation, it is necessary that continuity of fluxes and state vectors be maintained across the interface. To accomplish this, quantities at the mid-side node on the interface are set to the average of the two endpoint values. This makes quantities on both sides of the interface linear, guaranteeing continuity.

Alternatively, one can re-derive the finite element formulation for the case in which some of the interpolation functions are piecewise linear. This would have the advantage of not reducing the resolution along the interface, an important consideration if the interfaces are not aligned with the physical discontinuity. Time constraints prevented the implementation of this method at this time.

5.1 Examples

Two examples are presented to illustrate the use of adaptive degeneration, The first, $M_\infty = 0.675$ flow over a 10% circular bump, demonstrates the ability of the adaptive degeneration to remove shock oscillations. The second example, $M_\infty = 2$ flow in a scramjet inlet, demonstrates the ability of the degenerated elements to improve the robustness of the biquadratic algorithm.

The $M_\infty = 0.675$ bump example was computed using 300 biquadratic elements. Mach number contours are shown in Fig. 15. Note the formation of a normal shock on the bump. The significant overshoots and undershoots produced by the shock are visible in Fig. 16, which shows Mach number on the bump. Note also the odd-even mode propagating upstream of the shock. After degenerating 8 of the biquadratic elements into the equivalent of 32 bilinear elements, the oscillations and overshoots are significantly reduced (Fig. 17).

The second example, $M_\infty = 2.0$ scramjet inlet flow, is interesting because at this low Mach number the inlet unstarts, so a strong bow shock stands in front of the inlet. Since the flow in the inlet chokes, the location of the bow shock is determined by the inlet exit pressure. For this example, the exit pressure was set (somewhat arbitrarily) to 3.5 times the free-stream pressure, which corresponds to the pressure jump through a Mach 1.77 normal shock. In order to capture this bow shock, the grid of Fig. 4 was extended to include the region ahead of the inlet. The extension took about 1 hour of the first author's time to generate, further indicating the flexibility of the unstructured mesh approach. Figure 18 shows the Mach number contours for the solution. The interfaces between the biquadratic and bilinear regions are shown as dotted lines. The bow shock is captured clearly, as is the flow in the center passage. Note the presence of the the normal shock at $x = 2.67$, and observe the slip lines off the strut trailing edges. A plot of pressure and density at the exit plane (Fig. 19) clearly shows that the feature off the trailing edge of the struts is a slip line and not a shock. A slice through the inlet at $y = 0$ (Fig. 20) shows the bow shock and normal shock after the throat. Note that $M = 1$ at the throat ($x = 2.16$) as expected. This case cannot be run with biquadratic elements alone, because the initial transient in the solution produces such strong overshoots at the shocks that the calculation diverges. The final, converged solution also requires the linear elements around the bow shock to prevent divergence. The final grid used 602 biquadratic elements and 208 bilinear elements.

6 Three Dimensional Results

The extension of the 2-D, bilinear code to three dimensions is straightforward, involving almost no changes to the basic solver structure. The extension of the adaptation data structures, however, is far more complex. The computations involved in calculating the residual for the Galerkin method in 3-D are quite involved, and since the 2-D results showed little difference between the Galerkin and cell-vertex methods, the cell-vertex method was chosen for residual computations. Smoothing and boundary conditions are done as in the 2-D solver, with the addition of a second tangential velocity to the characteristics of Eq. 16.

To verify the three dimensional code, the test problem of $M_\infty = 2.5$ flow in a converging channel with both the y and z walls sloped at 10° was computed on a 30x30x30 grid. The geometry for this case is shown in Fig. 5. The double wedge introduces regions in which the flow may have passed through zero, one or two shocks. In addition, there are other kinds of interactions present (discussed fully by Kutler [19]). Figure 21 presents the density on a slice perpendicular to the flow direction. The shocks can be seen clearly, and in the region where the flow has passed through both shocks (the upper right corner), the results of the interactions are visible. In particular, note the bending of the shocks as they pass through each other and interact. This behavior is more pronounced as the shock strength increases. Figure 22 shows the Mach number on the plane $y = 0.5$. Here, the bending and interaction is clearly visible.

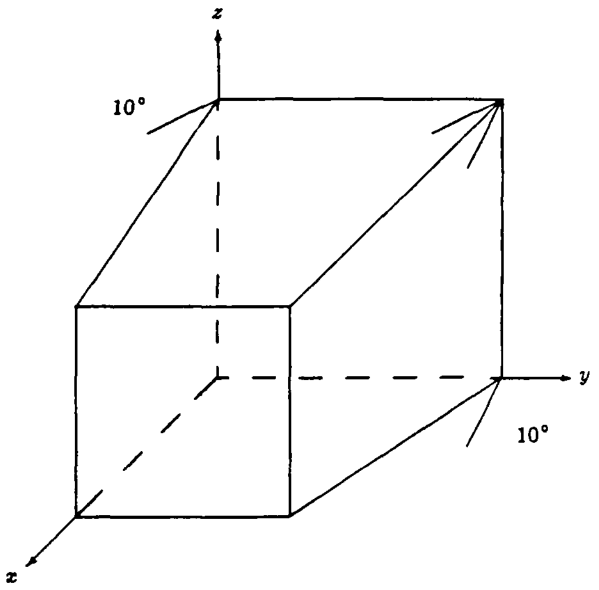


Figure 5: Geometry for 10° Double Wedge Flow

6.1 Adaptation Criteria

In order to decide which cells to divide or undivide, one needs to define some sort of adaptation parameter. There is a large amount of literature indicating possible choices for this adaptation parameter [20,21,22]. In this paper an indicator designed to capture shocks based on a scaled second difference of density is used. The second-difference based parameter is computed as follows:

1. Compute a nodal second difference of density.
2. Compute a nodal first difference as follows: elements, the following is used:

$$\text{First Difference}_i = \frac{1}{n_i} \sum_j |\rho_i - \rho_j|, \qquad (19)$$

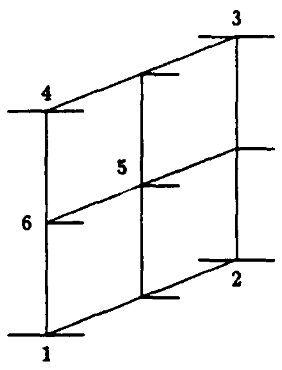


Figure 6: Cutaway View of Three-Dimensional Interface

where j ranges over all nodes in all elements containing node i, and n_i is the number of elements containing node i.

3. Compute an average density at the nodes.

4. Compute a nodal adaptation switch:

$$A_n = \frac{|\text{Second Difference}|}{|\text{First Difference}| + \epsilon \times \text{Average}}, \qquad (20)$$

where ϵ is a small parameter (of order 0.05) added to "smooth" the switch in smooth portions of the flow. Without it, small oscillations in the flow tend to produce large values of the switch.

5. Distribute the nodal switch to the elements.

6. If the switch is greater than some value (usually around 0.15), divide; if it is less than some value (usually around 0.05), undivide.

This indicator is based on results from interpolation theory which indicate that the interpolation error in a linear scheme is proportional to the second derivative of the quantity being interpolated. The first difference is used as a "fudge factor" to reduce the sensitivity of the switch to greatly differing shock strengths. Without the first difference scaling, the stronger shocks in the flow tend to receive most of the adaptation, while the weaker ones receive very little. Other advantages of this indicator are explained by Löhner [21].

6.2 Interface Treatment

The presence of embedding results in interfaces between the coarse and fine regions of the grid. In three dimensions there are two types of interface nodes: mid-face nodes and mid-edge nodes. Figure 6 shows a cutaway view of the interface between a fine (to the right) and coarse region (to the left) in three dimensions. In our treatment, the fluxes and state vectors at a mid-edge node (node 6) are the averages of the fluxes and state vectors at the ends of the edge (nodes 1 and 4 in the figure). The fluxes and state vectors at the mid-face node (node 5) are the averages of the fluxes and state vectors at the corner nodes (nodes 1-4). This treatment is easily vectorised, and the total time required to treat the interfaces is negligible.

6.3 Example

A three-dimensional scramjet test case with $M_\infty = 5$ was computed to demonstrate the 3-D adaptive algorithm. The geometry of the three-dimensional inlet is shown in Fig. 23, and is based on the two-dimensional geometry above. The leading edge sweep angle is 30°. A slice at $z = 0$ is identical to the two-dimensional geometry, and the slice at $z = 1$ has the portion of the struts and compression surfaces forward of the throat extended to give a 30° leading edge sweep. This weakens the compressions near $z = 1$, and forces the flow to turn down. Although the actual inlet proposed for the NASP project has a cowl plate which extends back from the throat, for the purposes of this study the cowl plate will be assumed to extend to the inlet mouth. For this case, contours of density at three z locations are shown in Fig. 25. Compare the 3-D contours with the 2-D, $M_\infty = 5$ density contours shown in Fig. 24, and note the differences introduced by the third dimension. Unfortunately, due to problems with the adaptation indicator, the side passages did not receive enough adaptation in the 3-D case, but the central passage indicates many of the important differences. At $z = 0.5$ and $z = 0.87$, note that the strut leading edge shocks reflect multiple times in the throat area, resulting in a significantly different flow in the expansion region. Also note the differences in the slip line positions for the three z slices. In the $z = 0.13$ slice, the slip line and the trailing edge shock are nearly on top of one another, while at $z = 0.87$ they are quite distinct. Figure 26 shows a density slice at $y = 0$. Features to note here are the coalescences of the multiple reflections in the throat area near $x = 1.2$, $z = 0.2$, and also a very faint, 3-D reflection extending from about $x = 1$, $z = 0$ to about $x = 2$, $z = 1$, due to the turning produced by the 30° strut sweep. This case was computed adaptively on the IBM 3090 at Cornell, had 188692 elements on the final grid and took over 12 hours of CPU time. Unfortunately, the mesh is not fine enough to capture many of the interesting interactions. If one more level of embedding is used, over 1.3 million elements would be needed, requiring about 480 megabytes of memory and about 1 week of single-processor IBM 3090 CPU time.

7 Conclusions and Further Work

The use of biquadratic finite elements in the solution of the Euler equations can result in significant computational cost reductions for many problems. The elements do show some oscillation near flow discontinuities, but these effects are relatively minor in many problems. For the times when oscillations are not acceptable, an adaptive degeneration of a biquadratic element into 4 bilinear elements proves quite helpful.

Further work is required on the adaptive degeneration concept. Two areas for research are improving the degeneration criterion and fixing the interface formulation to allow piecewise-linear interfaces. In addition the flux-corrected transport concept could be implemented as an alternative to degeneration.

The extension of the cell-vertex method to three dimensions was demonstrated, and the utility of adaptation was briefly explored. We consider the results reported here to be preliminary, as there is much work yet to be done on the choice of adaptation indicator, as well as work on a better artificial viscosity formulation. Further areas of research include the development of triquadratic elements. If the savings realized in 2-D apply to three dimensions, a single triquadratic element may well do the work of 64 trilinear elements.

Finally, work is in progress on adapting the finite element codes to massively parallel computers such as the Connection Machine. Preliminary results indicate that speeds on the order of 500 Megaflops are possible for the unstructured codes, and it is possible that speeds in excess of 1 Gigaflop are achievable.

8 Acknowledgements

This work was supported by the Air Force Office of Scientific Research under contract number AFOSR-87-0218, Dr. James Wilson, technical monitor. IBM 3090 time was provided by a grant from the Cornell National Supercomputer Facility. The first author wishes to acknowledge the support of the Fannie and John Hertz foundation.

References

[1] A. Jameson, "Successes and Challenges in Computational Aerodynamics," AIAA Paper 87-1184, 1987.

[2] R. Lõhner, K. Morgan, J. Peraire, and O. C. Zienkiewicz, "Finite Element Methods for High Speed Flows," AIAA Paper 85-1531, 1985.

[3] K. Bey, E. Thornton, P. Dechaumphai, and R. Ramakrishnan, "A New Finite Element Approach for Prediction of Aerothermal Loads–Progress in Inviscid Flow Computations," AIAA Paper 85-1533, 1985.

[4] R. Shapiro and E. Murman, "Adaptive Finite Element Methods for the Euler Equations," AIAA Paper 88-0034, January 1988.

[5] M. G. Hall, *Cell Vertex Schemes for the Solution of the Euler Equations*, Technical Memo Aero 2029, Royal Aircraft Establishment, March 1985.

[6] A. Jameson, "A Vertex Based Multigrid Algorithm for Three Dimensional Compressible Flow Calculations," In T. E. Tezduyar and T. J. R. Hughes, editors, *Numerical Methods for Compressible Flow: Finite Difference, Element and Volume Techniques*, American Society of Mechanical Engineers, New York, NY, 1986.

[7] K. Powell, *Vortical Solutions of the Conical Euler Equations*, PhD thesis, M.I.T., July 1987.

[8] R. L. Davis, *The Prediction of Compressible, Viscous Secondary Flow in Channel Passages*, PhD thesis, University of Connecticut, 1982.

[9] J. F. Dannenhoffer III, *Grid Adaptation for Complex Two-Dimensional Transonic Flows*, PhD thesis, M.I.T., August 1987.

[10] R. Lõhner, "FEM-FCT and Adaptive Refinement Schemes for Strongly Unsteady Flows," In *Proc. ASME Winter Annual Meeting*, Anaheim, California, December 1986.

[11] J.T. Oden, T. Strouboulis, P. Devloo, L. W. Spradley, and J. Price, "An Adaptive Finite Element Strategy for Complex Flow Problems," AIAA Paper 87-0557, January 1987.

[12] P. R. Eiseman, "Adaptive Grid Generation," *Computer Methods in Applied Mechanics and Engineering*, Vol. 64, 1987, pp. 321–376.

[13] J. Peraire, K. Morgan, J. Peiro, and O. C. Zienkiewicz, "An Adaptive Finite Element Method for High Speed Flows," AIAA Paper 87-0558, January 1987.

[14] J. Peraire, M. Vahdati, K. Morgan, and O. C. Zienkiewicz, "Adaptive Remeshing for Compressible Flow Computations," *Journal of Computational Physics*, Vol. 72, October 1987, pp. 449–466.

[15] R. A. Shapiro, *An Adaptive Finite Element Solution Algorithm for the Euler Equations*, PhD thesis, M.I.T., May 1988.

[16] R. Lõhner, K. Morgan, and O. C. Zienkiewicz, "The Solution of Non-linear Hyperbolic Equation Systems by the Finite Element Method," *Int. Journal for Numerical Methods in Fluids*, Vol. 4, 1984, pp. 1043–1063.

[17] A. Kumar, "Numerical Simulation of Scramjet Inlet Flow Fields," NASA Technical Paper 2517, 1986.

[18] S. T. Zalesak, "Fully Multidimensional Flux-Corrected Transport Algorithm for Fluids," *Journal of Computational Physics*, Vol. 31, 1979, pp. 335–362.

[19] P. Kutler, "Supersonic Flow in the Corner by Two Intersecting Wedges," *AIAA Journal*, Vol. 12, 1974, pp. 577–578.

[20] J. F. Dannenhoffer III and J. R. Baron, "Grid Adaptation for the 2-D Euler Equations," AIAA Paper 85-0484, January 1985.

[21] R. Löhner, "The Efficient Simulation of Strongly Unsteady Flows by the Finite Element Method," AIAA Paper 87-0555, January 1987.

[22] K. Powell, M. Beer, and G. Law, "An Adaptive Embedded Mesh Procedure for Leading-Edge Vortex Flows," AIAA Paper 89-0080, January 1989.

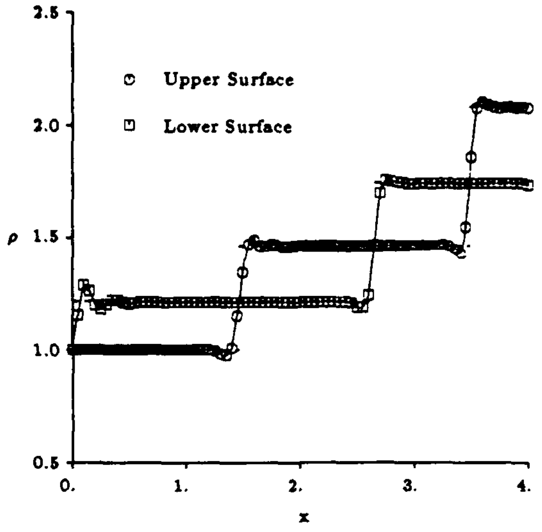


Figure 8: Surface Density, $M_\infty = 2$, 5° Converging Channel, Biquadratic Elements, 40x10 Grid

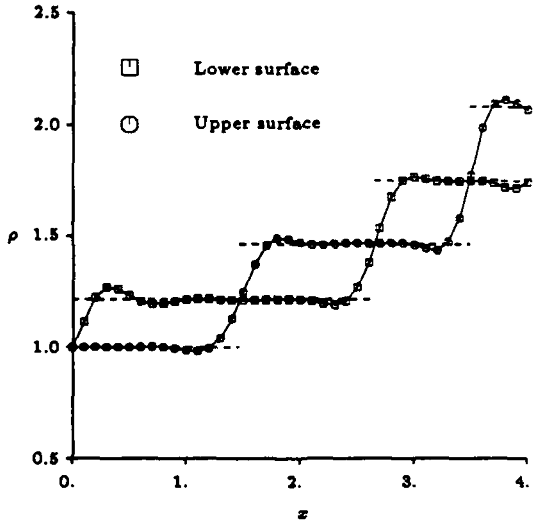


Figure 9: Surface Density, $M_\infty = 2$,5° Converging Channel, Bilinear Elements, 40x10 Grid

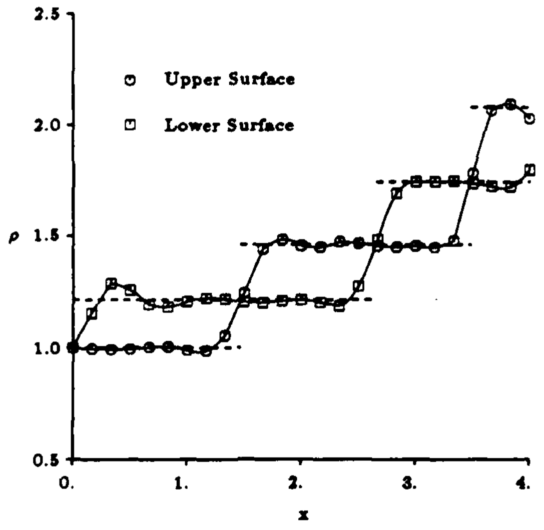


Figure 7: Surface Density, $M_\infty = 2$, 5° Converging Channel, Biquadratic Elements, 12x3 Grid

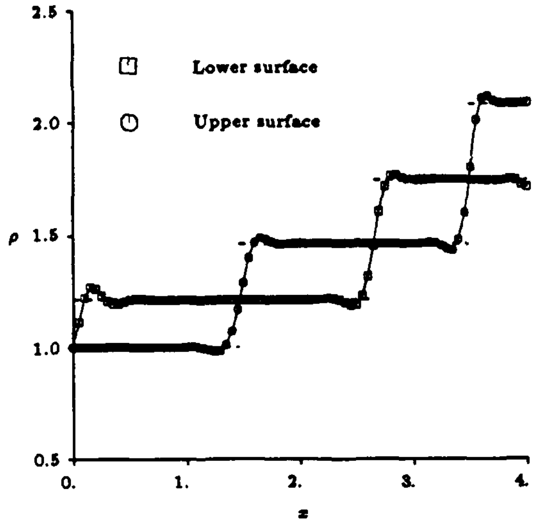


Figure 10: Surface Density, $M_\infty = 2,5°$ Converging Channel, Bilinear Elements, 80x20 Grid

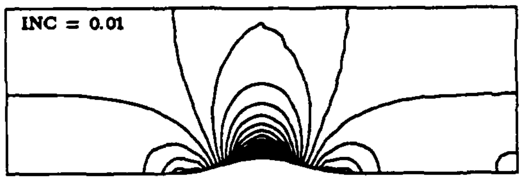


Figure 11: Density, 10% $\cos^2$ Bump, $M_\infty = 0.5$, 24x8 Grid, Biquadratic Elements

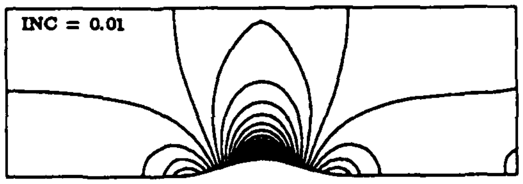


Figure 12: Density, 10% $\cos^2$ Bump, $M_\infty = 0.5$, 60x20 Grid, Bilinear Elements

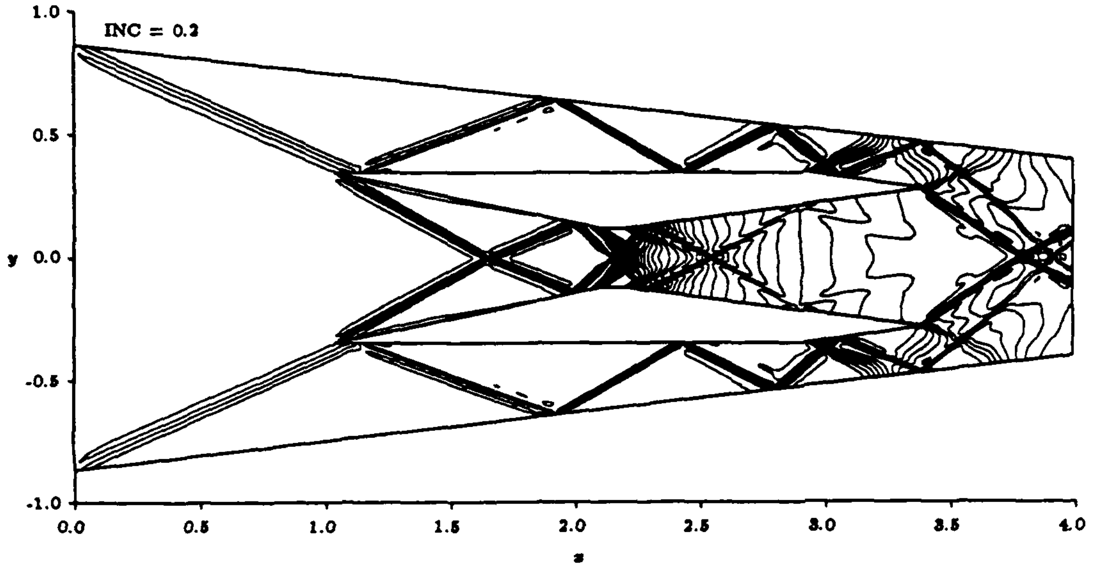


Figure 13: Density, $M_\infty = 3$, Bilinear Elements

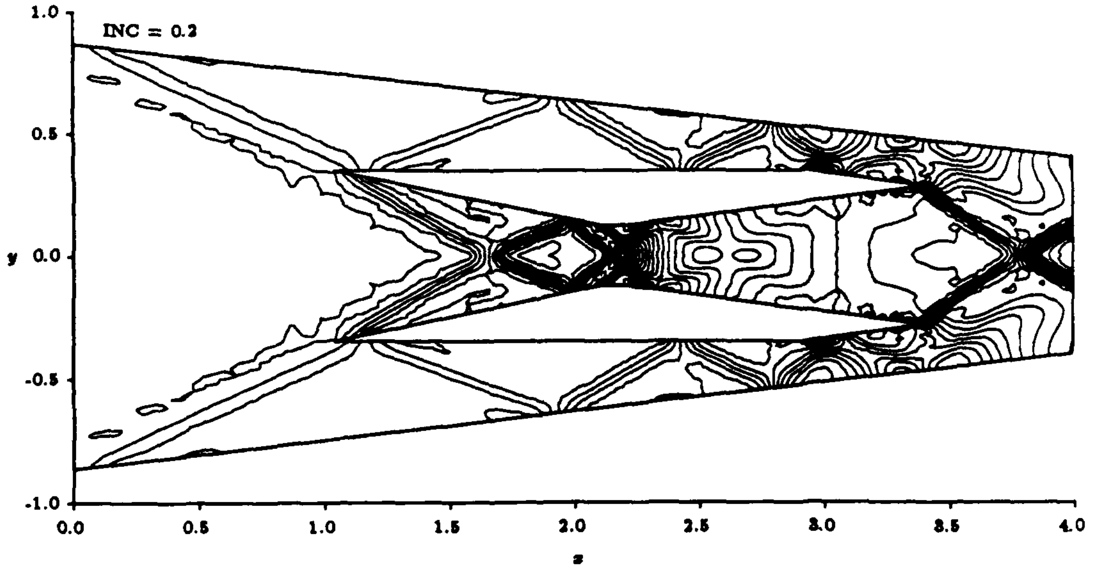


Figure 14: Density, $M_\infty = 3$, Biquadratic Elements

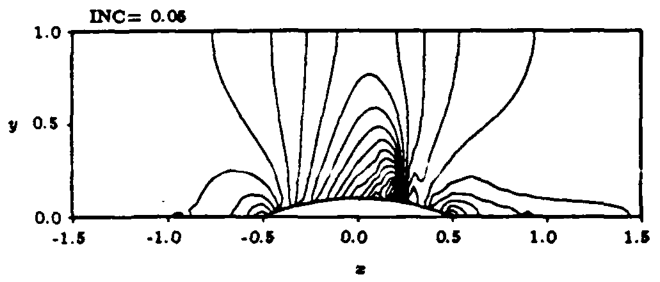


Figure 15: Mach Number, 10% Bump, $M_\infty = 0.675$, 300 Biquadratic Elements

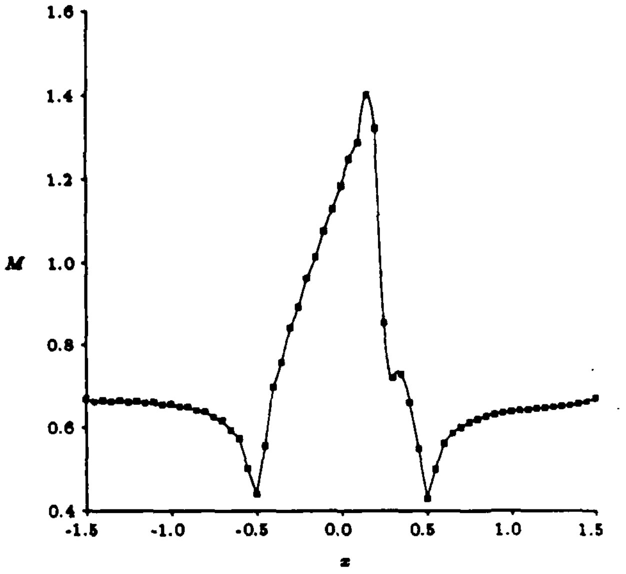


Figure 17: Mach Number, 10% Bump, $M_\infty = 0.675$, 292 Biquadratic Elements, 8 Bilinear Elements

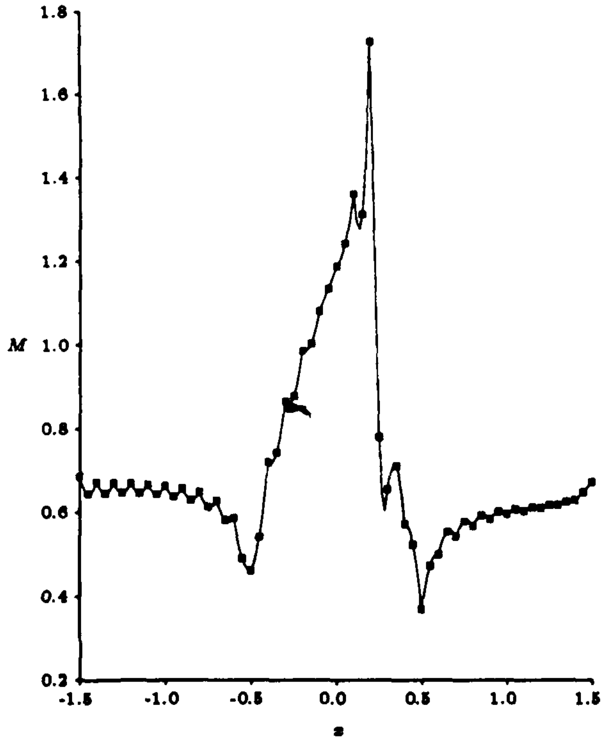


Figure 16: Surface Mach Number, 10% Bump, $M_\infty = 0.675$, 300 Biquadratic Elements

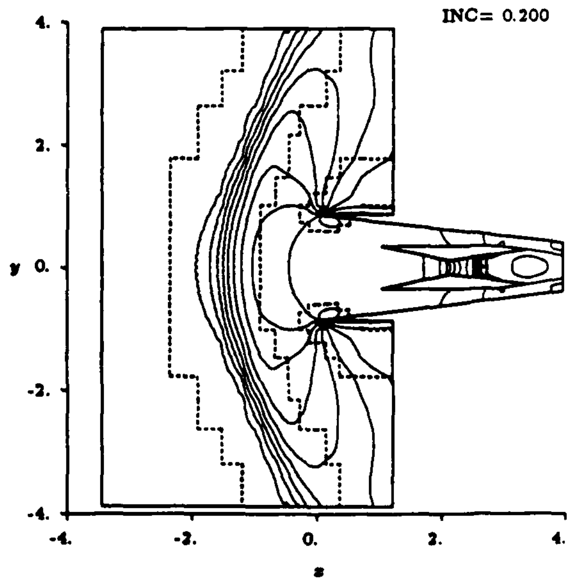


Figure 18: Mach Number, $M_\infty = 2$ Scramjet Inlet

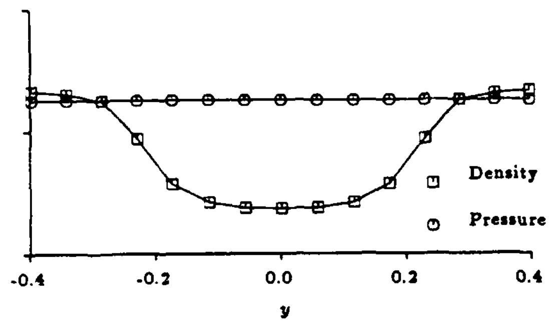


Figure 19: Pressure and Density at Exit Plane, $M_\infty = 2$ Scramjet Inlet

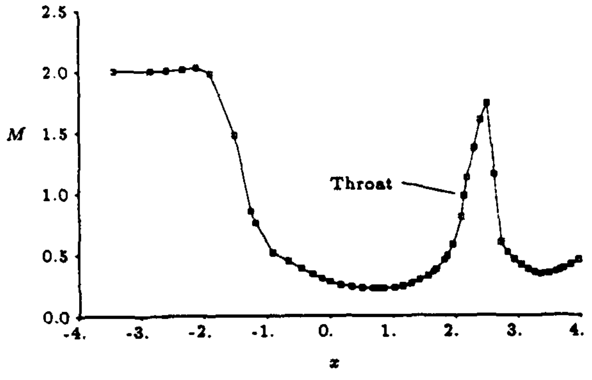


Figure 20: Mach Number at $y = 0$, $M_\infty = 2$ Scramjet Inlet

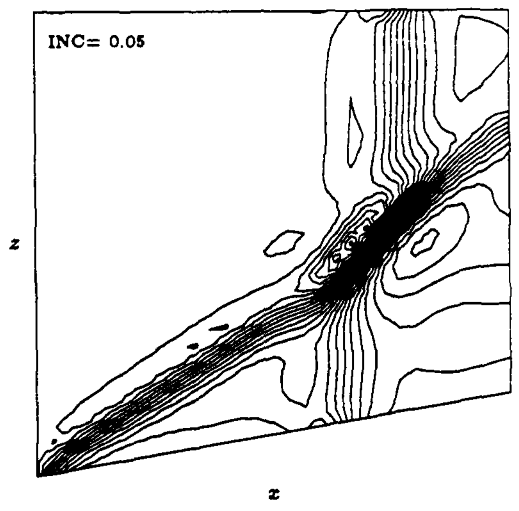


Figure 22: Mach Number, 10° Double Wedge, $M_\infty = 2.5$, X-Z Slice at Y = 0.5

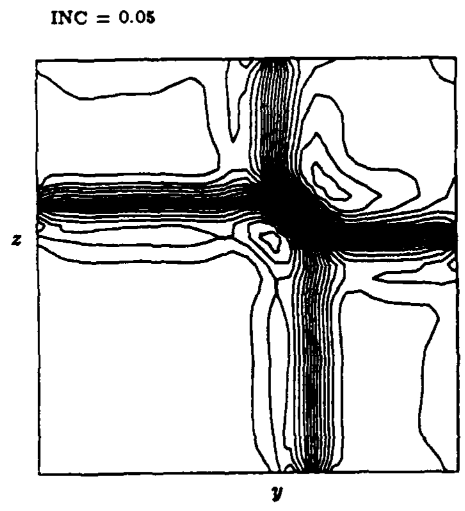


Figure 21: Density, 10° Double Wedge, $M_\infty = 2.5$, Y-Z Slice at X = 0.67

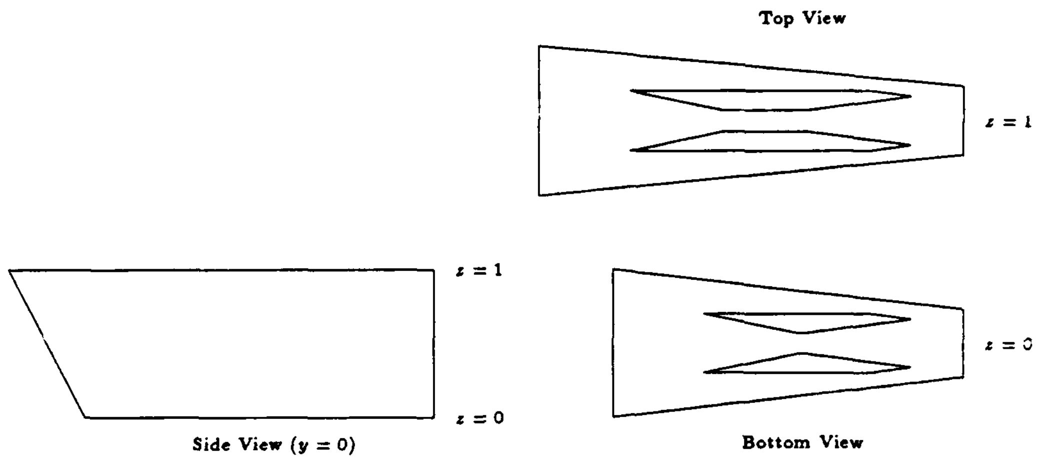


Figure 23: Geometry of Three-Dimensional Inlet

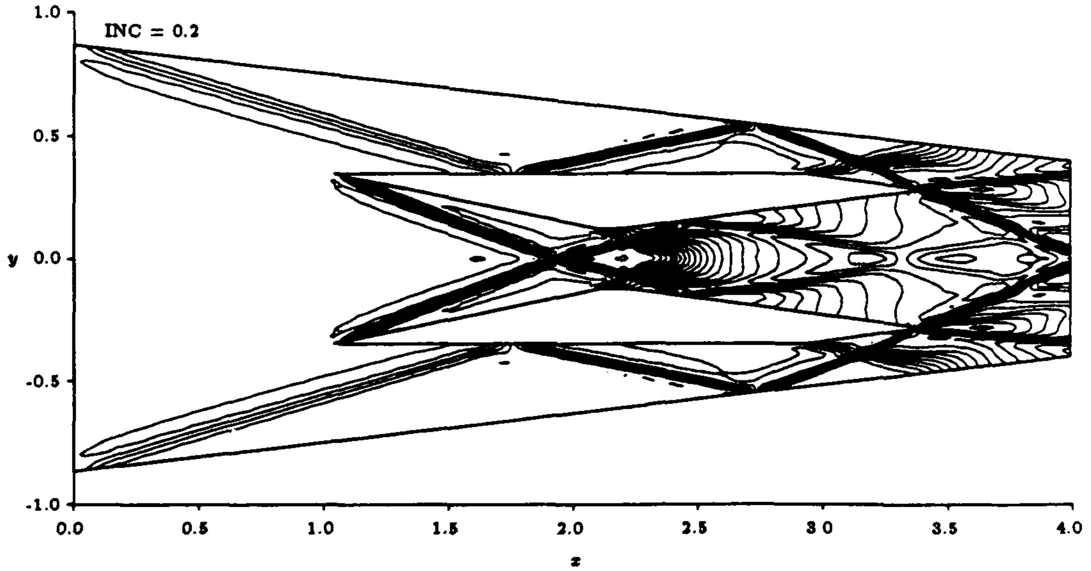


Figure 24: Density, $M_\infty = 5$, Bilinear Elements

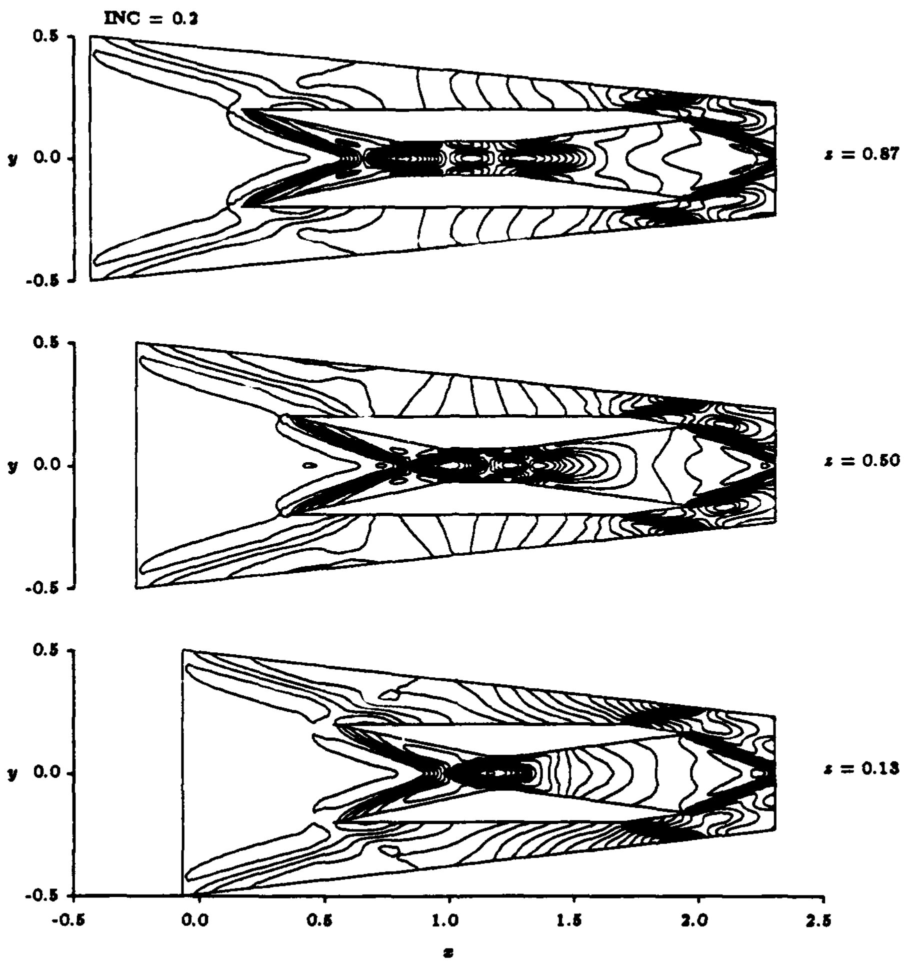


Figure 25: Density in $x - y$ Plane, $M_\infty = 5$, 3-D Case

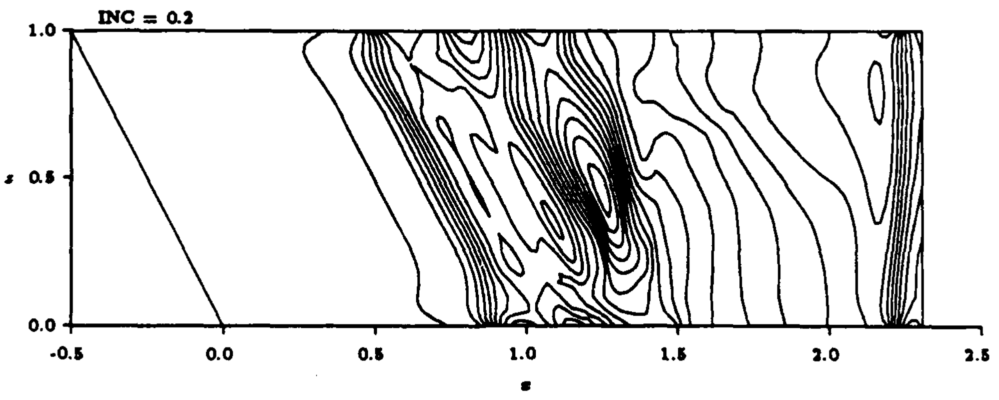


Figure 26: Density at $y = 0$, $M_\infty = 5$, 3-D Case

APPENDIX 12

Presented at AIAA 9th Computational Fluid Dynamics Conference, June 13-15, 1989.

AIAA 89-1941

Implementation of a 2D-Euler Solver using Id World, a Parallel Programming Environment

Alexandra M. Landsberg[*]

Earll M. Murman[†]

Computational Fluid Dynamics Laboratory

Department of Aeronautics and Astronautics

Massachusetts Institute of Technology

Cambridge, MA 02139

Abstract

A new parallel programming environment, Id World, is under development by MIT's Laboratory of Computer Science. This approach to parallel processing is distinct in that it employs a dataflow architecture and a functional programming language. In this paper, the application of Id World for CFD is explored by solving the 2D Euler equations using Jameson's scheme for the flow past a circular arc bump in a wind tunnel. Both subsonic and supersonic cases were run for a 17 × 5 grid. The parallel code was implemented in approximately 400 lines, equivalent to a sequential FORTRAN implementation. For one iteration, the average number of processors used in parallel was over 2000. Overall parallelism in the code as well as parallelism of specific steps of the method within the code are examined. Ease of implementation, determinancy, and memory allocation and deallocation are also discussed.

1 Introduction

Current state-of-the-art single-processor computers are pushing the upper bounds in computational speed. The hardware capabilities of these machines are limited due to single-processor technology approaching its theoretical limits. Therefore, in order to continue increasing computation speed, multiple processors in parallel must be used. This requires the development of parallel architectures, languages and algorithms. Due to the current dominance of von Neumann architectures and imperative languages, such as FORTRAN and C, the development of parallel von Neumann architectures and parallel imperative languages seems like a natural evolution [1]. A considerable amount of effort is being directed towards this goal.

However, another very distinct approach to the parallel programming problem is being researched in the Computation Structures Group at MIT's Laboratory of Computer Science. They have developed a parallel environment called Id World based on a dataflow architecture, a "Multiple-Instruction/Multiple-Data" architecture, and a functional programming language called Id. Although Id is designed for dataflow machines, it is not limited to this class of architectures. The Computational Fluid Dynamics Laboratory in MIT's Department of Aeronautics and Astronautics is also interested in studying parallel architectures and algorithms for CFD applications. In this paper we will report the results of implementing a 2D Euler solver in Id World. The algorithm, given to students in the advanced CFD course, intuitively possesses a high degree of parallelism. Through the use of Id World the amount of parallelism inherent in the algorithm is explored.

2 Algorithm Description

The algorithm implemented in Id solves the steady compressible Euler equations for the flow past a circular arc in a 2D channel using Jameson's scheme [2]. The compressible Euler equations in Cartesian coordinates can be written in vector form as

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0 \qquad (1)$$

[*]Research Assistant, Member AIAA

[†]Professor, Fellow AIAA

where the state vector U and the flux vectors F and G are defined by

$$\mathbf{U} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix} \quad \mathbf{F} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ \rho u H \end{pmatrix} \quad \mathbf{G} = \begin{pmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ \rho v H \end{pmatrix} . \qquad (2)$$

Assuming a perfect gas, with specific heat ratio γ, pressure and enthalpy are defined as

$$p = (\gamma - 1)\rho(E - \frac{1}{2}(u^2 + v^2)) \qquad (3)$$

$$H = E + \frac{p}{\rho} \qquad (4)$$

where ρ, u, v, E, p, and H are, respectively, the density, Cartesian components of velocity, total energy per unit mass, pressure and total enthalpy per unit mass. The Euler equations represent the conservation of mass, momentum and energy for an inviscid perfect gas.

The first step is to generate a computational grid representing the physical plane using an algebraic grid generator. The entire flowfield is then initialised to the freestream values and the boundary conditions are set. An outer loop is set up to iterate and solve the entire flowfield. Within this loop, the local time step, denoted as Δt, and artificial viscosity are calculated once and a four-stage Runge-Kutta method is then used to improve the solution. The Runge-Kutta method involves calculating the fluxes over the sides of the cells and then calculating the residuals. The residual of a cell is the discrete approximation of

$$-\frac{\partial \mathbf{F}}{\partial x} - \frac{\partial \mathbf{G}}{\partial y} = R_{i,j} \qquad (5)$$

where R_{ij} is termed the residual. The Runge-Kutta method is as follows:

$$\mathbf{U}_{ij}^{(0)} = \mathbf{U}_{ij}^{n}$$
$$\mathbf{U}_{ij}^{(1)} = \mathbf{U}_{ij}^{(0)} - \alpha_1 \Delta t_{ij} R(\mathbf{U}_{ij}^{(0)})$$
$$\mathbf{U}_{ij}^{(2)} = \mathbf{U}_{ij}^{(0)} - \alpha_2 \Delta t_{ij} R(\mathbf{U}_{ij}^{(1)})$$
$$\mathbf{U}_{ij}^{(3)} = \mathbf{U}_{ij}^{(0)} - \alpha_3 \Delta t_{ij} R(\mathbf{U}_{ij}^{(2)})$$
$$\mathbf{U}_{ij}^{(4)} = \mathbf{U}_{ij}^{(0)} - \alpha_4 \Delta t_{ij} R(\mathbf{U}_{ij}^{(3)})$$
$$\mathbf{U}_{ij}^{n+1} = \mathbf{U}_{ij}^{(4)}$$

with $\alpha_1 = \frac{1}{4}$, $\alpha_2 = \frac{1}{3}$, $\alpha_3 = \frac{1}{2}$, and $\alpha_4 = 1$. This scheme is of second order accuracy for a smooth mesh. It should be noted here that there are dependencies between stages. How these dependencies affect parallelism will be examined later. The inflow, outflow, and solid wall boundary conditions are imposed after each stage of the Runge-Kutta method. The convergence criterion is based on the variation in pressure between the previous iteration and the current iteration. Both a root-mean-square test and the maximum pressure difference is calculated. If the convergence criteria is not met, the outer loop is repeated until convergence. Calculations were done for subsonic and supersonic flow past a circular arc bump in a channel using a 17 × 5 grid.

3 Id World

In the engineering world FORTRAN is the dominant language. The usual approach to parallel programming is to annotate FORTRAN code with parallel constructs and then to compile the code for a parallel architecture. For some machines, such as the Alliant, compilers exist which automatically parallelise FORTRAN code, if it is suitably written for the particular parallel strategy. Often, the *determinacy* of the program is left to the programmer; i.e., the programmer must insert appropriate synchronisations to ensure that for a given set of inputs, the output will always be the same. Debugging becomes extremely difficult when the program may produce different outputs for different machine configurations and/or scheduling policies. In addition, this behavior may not be immediately obvious. What further complicates these time-dependent errors is that they may not even be reproducible in a debugger [3]. A fundamental question arises: "What is the correct output for a given set of inputs; i.e., is the parallel program executing correctly?"

Id World was designed to overcome the problem of determinacy and easily enable one to study the parallel behavior of algorithms. Id World consists of the parallel programming language Id, an Id compiler, and GITA (Graph Interpreter for the Tagged-Token Architecture), an emulation of the MIT Tagged-Token Dataflow Architecture (TTDA). Monsoon is a hardware implementation of the TTDA with extensions.

In Id World, programs are written in the parallel programming language Id. One then compiles Id programs using the Id compiler. The compiler then produces dataflow graphs. Next, the dataflow graphs are executed on a tagged-token dataflow architecture. Either GITA can be used to execute the dataflow graphs or Monsoon can be used. Once a program has finished execution, the parallelism of the algorithm can be examined from the parallelism profiles.

3.1 Id

Id is a general-purpose high-level parallel language with data structures suitable for scientific applications and symbolic computations (AI applications). Id is a functional language. In a functional language, a data structure can never be overwritten – "no side effects". Therefore updating is accomplished through the copying of data structures. This will quickly result in memory problems, an issue which will be discussed in more detail later. For all purely functional languages, no side effects guarantees determinacy.

Another feature of Id is that the parallelism is *implicit* in the operational semantics of the language. The programmer does not explicitly break up a task into parallel components, and therefore does not worry about synchronisation. The amount of parallelism in a program is determined by the data dependencies in the algorithm. The Id compiler determines only the essential data dependencies in the algorithm, thus relieving the programmer of the task [3].

In addition, Id has dynamic storage allocation which enables extra storage to be dynamically allocated for more parallelism. Id also supports fine-grain parallelism. The more finely the Id compiler divides a program into tasks, the greater the opportunity for parallel execution. However, the overhead cost of synchronization must be minimised in order to gain in performance [4]. Instead of addressing this problem at the software level, this problem is addressed and solved at the hardware level. The Monsoon processor accomplishes low synchonisation cost.

Id is a layered language, being purely functional at its core. Id is extended with I-structures as a parallel data-structuring facility. An I-structure is a data structure, such as an array, that is immemdiately allocated memory, but does not need to be written immediately. An I-structure can never be overwritten [5]. In comparison, in a purely functional language a data structure is both allocated memory and written immediately, and then can never be overwritten.

An important control structure for numerical computations is an array comprehension. Array comprehensions allow an array or matrix to be created and initialised. Array comprehensions initialise arrays that may either be purely functional data structures or I-structures [6]. The following array a must be an I-structure.

```
a = {1d_array (1,4)
     | [1] = 2.0
     | [j] = a[j-1] + 2 || j <- 2 to 4 }
```

The above array comprehension creates and initialises an array with bounds from 1 to 4. This array is bound to the variable a. The first symbol of the array comprehension, in this case array, specifies the type of the array, i.e. 1D, 2D, etc., while the second component specifies the bounds of each dimension. This is followed by a set of mutually exclusive clauses, where each clause is separated by a |, that initialise the array entries. The first clause fills the first entry of the array with the value 2.0. The second clause fills entries 2 through 4 of the array with a[j-1] + 2. The symbol || indicates the range over which j is to be looped.

A CFD application of array comprehensions is filling all the interior cells of a grid, and then specifying the boundary conditions at the inlet and outlet. Therefore, one matrix is created containing all the information for the grid [7]. The following is a typical application of array comprehensions.

```
U = {2d_array ((0,imax),(0,jmax))
     | [i,j] = interior (i,j) || i <- 1 to imax-1
                               & j <- 0 to jmax
     | [0,j] = inlet (j) || j <- 0 to jmax
     | [imax,j] = outlet (j) || j <- 0 to jmax}
```

The array comprehension creates and initializes a matrix with bounds 0 to imax for the first dimension and bounds 0 to jmax for the second dimension. The matrix is bound to the variable U. The first clause fills the i,j entries of the matrix, where i goes from 1 to imax-1 and j goes from 0 to jmax, with the values returned from the function interior. The values returned from interior may be any data structure. For example, if a vector is returned from interior then each entry of U contains a vector. Similarly, the inlet and outlet cells of the grid are filled by the values returned from the functions inlet and outlet, respectively. This is illustrated in Figure 1.

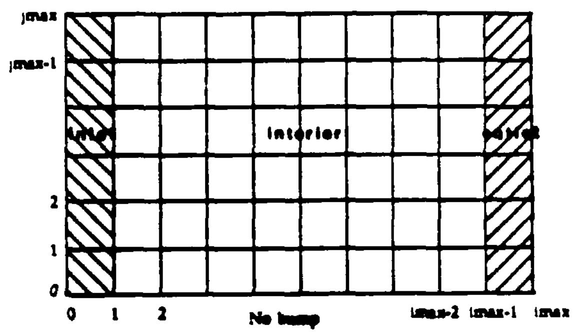


Figure 1: Initialisation of U

In addition to array comprehensions, Id has a large library of array functions necessary for numerical computations. The array library is implemented to achieve maximum parallelism of the functions.

Another important data structure in Id is a "tuple". A tuple is a data structure that represents a collection of objects. An n-tuple contains n objects. The primary use of tuples in this application was for array/matrix bounds and for returning multiple values from a function. A function can return only one value. A tuple can be used to package results when a function needs to return multiple values. The following example illustrates the use of tuples.

```
bounds = ((0,imax),(0,jmax));

def calc_pressure p_infinity =
  {def pressure (i,j) = p_infinity[i,j] ;
   in
     make_matrix bounds pressure};
```

In this example, bounds is a 2-tuple where each element of the tuple is a 2-tuple that represents the bounds of the matrix. The key word def indicates a function definition. The function calc_pressure is creating and initializing a matrix with bounds bounds. Calc_pressure takes one argument, p_infinity, which in this case is a matrix. The body of the function is expressed as a "block expression". A block expression contains a set of statements at the end of which is the key word in. The expression following in represents the value being returned by the block. In this example, a matrix is being created and initialized by make_matrix which is a function in the array library that takes two arguments, the bounds of the matrix and a function. The entries of the matrix are being filled by the values returned from the function pressure. Also note within the block expression that the function pressure takes one argument, the tuple (i,j), which will be bound to each pair of indices of the matrix being created. In our application, we extensively use array comprehensions and make_matrix, thus approximately 30% of the data structures we use are tuples.

3.2 Dataflow Graphs

A dataflow graph is the parallel machine language of dataflow machines. A node in a dataflow graph is an instruction which executes only when the operands it requires are available. The arcs in the graph represent the essential data dependencies among instructions - "data-driven". A dataflow graph imposes only a *partial order* of execution; all execution orders which obey the data dependencies in the graph yield the same result. This is the source of parallelism in dataflow graphs – a valid execution is to process in parallel all instructions which have their required operands [8]. For the following block expression (Example 4), the dataflow graph is shown in Figure 2.

```
{ x = 2*3 ;
  y = x*x + 3*5 ;
  z = 1 + 2 ;
 in
   (x+y)*(y-z) };
```

In contrast, a sequential processor imposes a *total order* on all operands within a task.

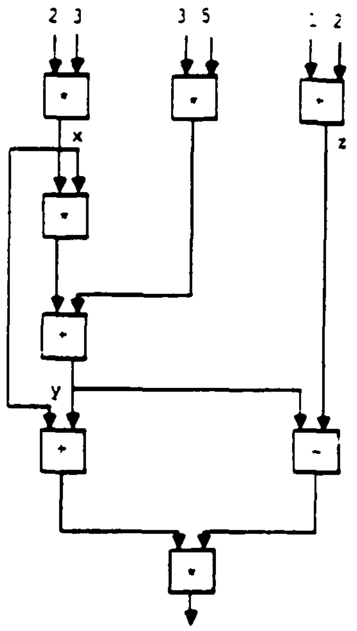


Figure 2: Dataflow Graph

If Example 4 were in a loop from 1 to 4 then there would 4 activations of that block expression. Now examining just the first line of that block expression, x=2*3, the operands are 2 and 3 while the instruction is *. Each operand to the instruction * has a tag that identifies to which activation the operand belongs. The tokens destined for each instance of * are distinguished by differing tags. Therefore, for this line of code, within each activation there are two tokens destined for the instruction * which must have matching tags.

A combination of hardware and compiler disciplines guarantee that only two tokens will ever have the same tag for a two operand instruction. This same feature guarantees the determinancy of the program.

3.3 Tagged-Token Dataflow Architecture

The TTDA is a data-driven machine which directly executes dataflow graphs. It is called a tagged-token dataflow architecture because each operand to an instruction is tagged with an identifier. A token is an operand and its tag. Therefore, a two operand instruction has two tokens that must have matching tags. Moreover, tags distinguish tokens from different "activations" [8]. For example, all of the iterations of a loop potentially may be operating in parallel and therefore represent different activations.

3.4 Hardware

GITA is a software system that interprets dataflow graphs. GITA is run on a Multiprocessor Emulation Facility (MEF). The MEF consists of 32 TI Explorer Lisp Machines linked together with a communcation network which emulate parallel processing. The MEF is used primarily for code development and debugging. GITA performance on one TI Explorer is only 1,000 instruction per second which is prohibitively slow to run any large applications. The advantage of networking is that performance and memory increase proportionally with the number of machines used. However, actual execution on the machines is asynchronous.

The Monsoon is a 256 processor dataflow machine with a TI Explorer as its front end. The Monsoon is a TTDA implementation with extensions for efficient matching of tokens. Currently, one prototype processor has been built. The performance for one processor is 8 MIPS, while an estimate for 256 processors is 1,000 MIPS (assuming 50% processor utilization).

3.5 Parallelism Profiles

The parallelism profile of an algorithm is the number of *processors versus computation timesteps* (*there is no connection between a computation timestep and any unit of real time*). For the previous code block, the parallelism profile is shown in Figure 3.

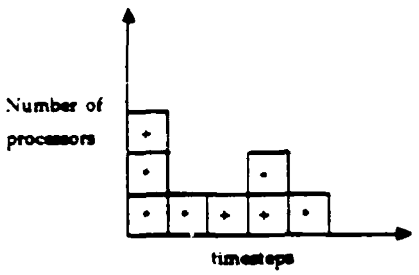


Figure 3: Parallelism Profile

A program can be run in one of two modes: *Idealised Mode* or *Finite Processor Mode*. Idealised Mode assumes the following: every instruction executes in a unit timestep, there is no communication latency from one instruction to its successor, and every instruction executes at the earliest possible timestep, i.e., as soon as its input data is ready, and finally, an infinite number of processors is assumed. Finite Processor Mode assumes: every instruction executes in a unit timestep, the result of an instruction takes m timesteps to reach its successor instruction – communication latency, and that there are n processors (limited from 1 to 1,000) [3].

4 Discussion of Results

4.1 Ease of Implementation

Although Id is a functional language and does not allow overwriting of data structures, a FORTRAN programmer can easily learn how to program in Id. This algorithm was implemented by the first author, who had to learn both the Euler algorithm and Id. The algorithm was implemented in approximately 400 lines of Id code. This same algorithm implemented in a sequential FORTRAN takes approximately 300-600 lines of code based on the style of the programmer.

The key to programming in Id is understanding how to write functions. Unlike most imperative languages, arguments and results of functions can be arbitrarily complex types: arrays, trees, lists, ... even functions [5]. For example, an array may contain a list in each cell and a function may return a function. This is important to remember in the design and implementation of the algorithm.

4.2 Design and Implementation

The aim of this project was to design a high level implementation of the algorithm. The bulk of the computations are in the four-stage Runge-Kutta loop, specifically in the *residuals* calculations. The algorithm specifies the calculations primarily as vector manipulations, such as adding and subtracting vectors. Therefore, a high level representation would work with the vectors U, F, and G directly. This is easily accomplished since data structures can be arbitrarily complex. Thus, instead of representing U, F, and G in a 3D array, i.e., $U(i,j,k)$ where (i,j) is the cell coordinate and k is the coordinate in the vector, represent U, F, and G as vectors in a 2D array, where each cell contains the 3 vectors. The required vectors operations for our application were adding vectors, subtracting vectors and multiplication of a vector by a scalar. The array library could have been used to perform the required vector operations; however, in this case it was easier to define these functions.

4.3 Parallelism Statistics

Two sets of parallelism statistics were collected. First, the Arithmetic Logical Units (ALU) Operations profile shows the total number of instructions performed in parallel per time step. The second profile collected is the number of floating point operations performed in parallel per time step.

Both subsonic and supersonic cases were run; however, all statistics shown are for supersonic flow on a 17×5 grid. The amount of parallelism in the algorithm can be assessed by examining only a small number of iterations since the same calculations are performed in each iteration after the first. Therefore, all parallelism profiles are for three iterations (this work was performed using GITA on the MEF, which as stated above is prohibitively slow). Finally, all profiles were run using Idealised Mode, thus showing the maximum amount of parallelism possible.

The ALU operations and floating point operations profiles are shown in Figure 4 and Figure 5, respectively. The ALU operations profile shows that on average over 1,500 instructions can be performed in parallel per iteration. The floating point operations profile also shows a high degree of parallelism. The dips in the parallelism profiles show the data dependencies between iterations. Within one iteration there are no dips in the parallelism profiles; however, a four-stage Runge-Kutta loop is executing. As shown before, the

Runge-Kutta loop has data dependencies between stages and intuitively the method seems sequential. However, Id is able to determine the maximum amount of parallelism in the algorithm since parallelism is only inhibited by data dependencies, and as can be seen the Runge-Kutta loop is highly parallel.

4.4 Loop Bounding and Parallelism Profiles

From the ALU operations profile, the first iteration uses a peak of over 13,000 processors while the later iterations use 1,500 processors on average. This is due to all of the loop information "unfolding", i.e., all of the loops begin to execute in parallel until they reach a data dependency. This is clearly undesirable since this kind of parallelism will consume all available resources. Therefore, we want to "bound" the loop from unfolding too many iterations at once. In this application, only two iterations at a time need to be unfolded, the current and previous iterations. The previous iteration is needed to update the current iteration. Figure 6 and Figure 7 show the ALU operations and floating point operations profiles with loop bounding. As can be seen from these figures, each iteration including the first is now identical. A peak of 10,000 processors can be used, while an average of 2,000 processors is used per iteration. Thus, loop bounding has more evenly distributed the work over the processors.

4.5 I-Structure Storage

As mentioned previously, the updating of data structures is accomplished through copying. Every vector operation, such as adding two vectors, results in a new vector being created. This is needed to guarantee determinacy; however, this will clearly result in a memory problems very quickly. A profile of I-Structure versus timesteps was collected for the case with loop bounding since each iteration is the same. Therefore, the profile of I-Structure storage versus timesteps shows how many bytes of memory is being used for each iteration. As can be seen from Figure 8, each iteration requires slightly over 50 kbytes of memory. Since every data structure that is created can never be overwritten, these data structures remain in memory until execution is complete. The supersonic case requires approximately 400 iterations for convergence, at 50 kbytes an iteration, that would require 20 Mbytes of memory. Clearly, a facility to "release" old data structures, i.e. data structures that are no longer needed, is necessary.

Currently, the only means of releasing or reclaiming storage used by old data structures is by explicit deallocation. This is accomplished with the following syntax:

```
@release garbage_data_structure
```

@release can be placed anywhere within a function, but it will not be invoked until just before the function returns [6]. Therefore, within the function, @release will not cause errors because it is not possible for a computation within the function to access the garbage data structure after it is released. However, a problem arises when an @release is used within a function but external to that function there is some dependency on that released data structure. The following example clearly demonstrates the problem:

```
def add_vectors A B =
 {@release A,B ;
  in
   {array (1,4)
    | [j] = A[j] + B[j] || j <- 1 to 4}}

M = R[3];
T = add_vectors R S;
```

In this example, once the two vectors, A and B, are added in the function add_vectors, those vectors are released; however, the computation to initialize M and T are running in parallel. If the call to add_vectors finishes before R[3] is accessed then R will have been released and an error will result because an attempt will be made to access a garbage data structure; however, if R[3] is accessed before the call to add_vectors finishes then no error will occur. Thus, @release can introduce non-determinism into a program [9].

4.6 Sources of Excess I-Structures

With this in mind, a key question arises: "Where is this "garbage" coming from and how easily and effectively can it be explicitly deallocated?"

Garbage is created from three primary sources: vector operations, temporary vectors within functions, and tuples. Vector operations account for approximately 40% of all garbage, temporary vectors within functions account for 25% and tuples account for 30%. The remaining 5% is categorised as "other".

The task of minimising the number of I-structures being created and subsequent deallocation of old I-structures is not extremely difficult, but can be time-consuming. More importantly, in this application explicit sequentialization had to be imposed in two functions in order to release old data structures. This is a major disadvantage of @release.

After implementing the necessary reduction and deallocation options for data structures, the resulting I-Structure storage profile is shown in Figure 9. The amount of garbage is only 1000 bytes, 2% of the original amount. Therefore, running the supersonic case to convergence would require 500 kbytes of memory, while the subsonic case which requires approximately 3,000 iterations for convergence would still need 3 Mbytes of memory. The ALU operations and floating point operations profiles are shown in Figure 10 and Figure 11, respectively. These profiles show a significant amount of sequentialization occurs due to the releasing old data structures. In fact, the number of computation timesteps is almost double the number required for the case with loop bounding . Therefore, although explicit deallocation is very effective, the loss of parallelism is highly undesirable. In addition, for CFD applications the amount of garbage must essentially be zero for practical applications.

4.7 Garbage Collectors

What is needed is a facility to automatically deallocate old data structures. LISP systems use a "garbage collector" to deallocate old data structures once a certain amount of memory has been filled. Designing a garbage collector for Id to deallocate basic data structures such as arrays, lists, and tuples is not a difficult task. However, the Computation Structures Group is working on the more general problem of how to determine when an arbitrarily complex data structure can be released. Using a combination of explicit deallocation and a garbage collector for Id would efficiently deallocate *all* old data structures and impose much less sequentialization. Thus, a garbage collector is essential for running CFD applications.

5 Summary

A 2D Euler solver for the flow over a circular arc bump was implemented in Id World. Id World is a distinct approach to parallel processing that uses a functional language, Id, and a Tagged-Token Dataflow Architecture. Id is a powerful, general-purpose language with data structures suitable for numerical problems. Algorithms can easily be expressed with a high-level representation in Id. Vector manipulations are greatly facilitated by array comprehensions and the array library.

The TTDA, a "data-driven" machine, is inherently parallel since it directly executes dataflow graphs. The Monsoon is a hardware implementation of the TTDA with extensions that have made dataflow architectures practical. With the implementation of a garbage collector, practical CFD problems can be solved.

In conclusion, Id World is an excellent tool for analysing the implicit parallelism in algorithms and for developing new parallel algorithms.

6 Acknowledgements

This research was sponsored by the Air Force Office of Scientific Research under Grant AFOSR-87-0218 monitored by Dr. Len Sakell. The authors would like thank the Computation Structures Group, specifically Prof. Arvind, James Hicks, and Dr. Gregory Papadopoulos, for their invaluable assistance.

References

[1] Arvind and K. Ekanadham, *Future of Scientific Programming on Parallel Machines*, CSG Memo, Laboratory of Computer Science, Massachusetts Institute of Technology, February 1988.

[2] A. Jameson, *Solution of the Euler Equations for Two Dimensional Transonic Flow by a Multigrid Method*, Princeton University MAE Report No. 1613, June 1983.

[3] R. S. Nikhil, P. R. Fenstermacher, and J. E. Hicks, *Id World Reference Manual*, CSG Memo, Laboratory of Computer Science, Massachusetts Institute of Technology, August 1988.

[4] Arvind, D. E. Culler, and G. K. Maa, *Assessing the Benefits of Fine-grained Parallelism in Dataflow Machines*, CSG Memo, Laboratory of Computer Science, Massachusetts Institute of Technology, June 1988.

[5] Arvind, "Course notes from Dataflow Architectures and Languages," 1988.

[6] R. S. Nikhil, *Id (Version 88.0) Reference Manual*, CSG Memo, Laboratory of Computer Science, Massachusetts Institute of Technology, March 1988.

[7] K. Ekanadham and Arvind, *SIMPLE: Part I, An Exercise in Future Scientific Programming*, CSG Memo, Laboratory of Computer Science, Massachusetts Institute of Technology, July 1987.

[8] G. M. Papadopoulos, *Implementation of a General Purpose Dataflow Multiprocessor*, PhD thesis, M.I.T., December 1988.

[9] J. A. Gilson, Personal communication.

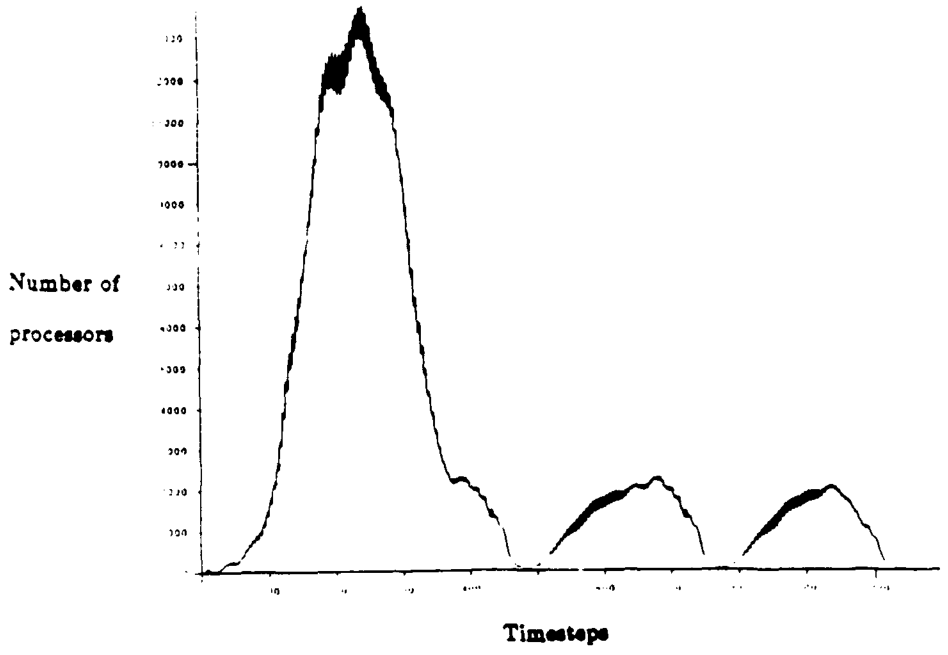


Figure 4: ALU Operations Profile

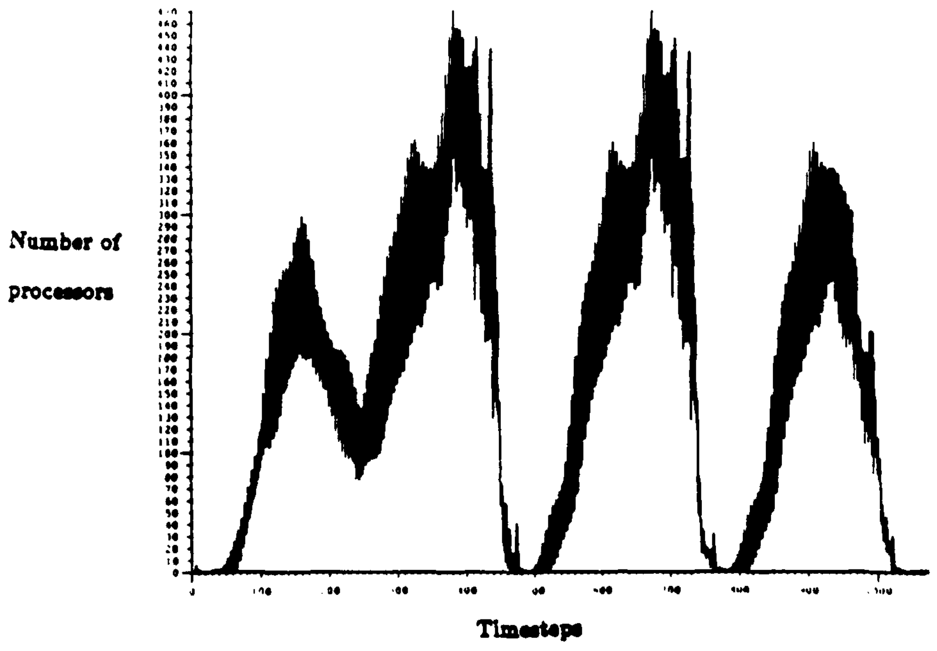


Figure 5: Floating Point Operations Profile

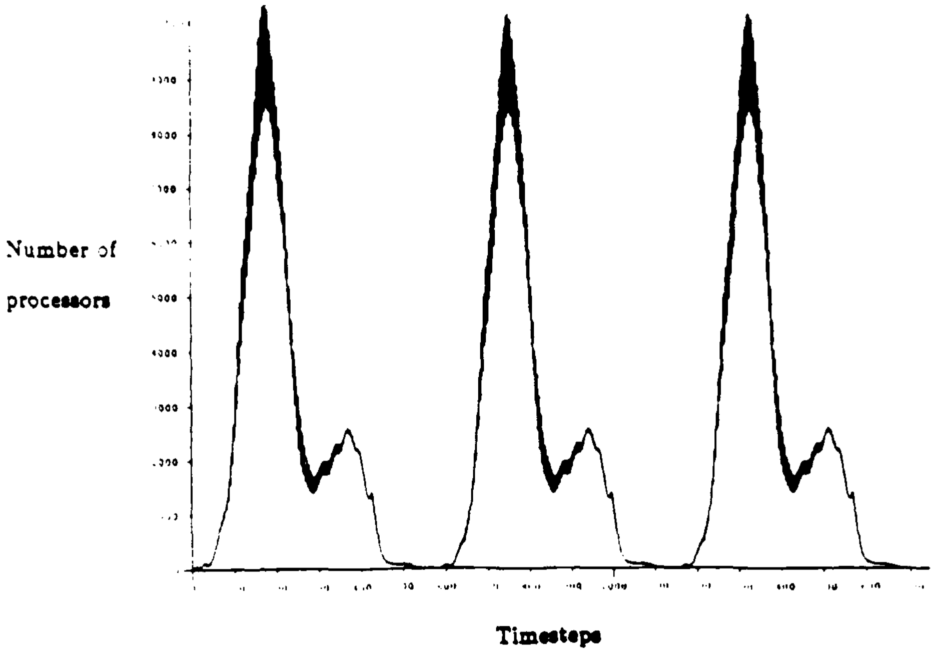


Figure 6: ALU Operations Profile with Loop Bounding

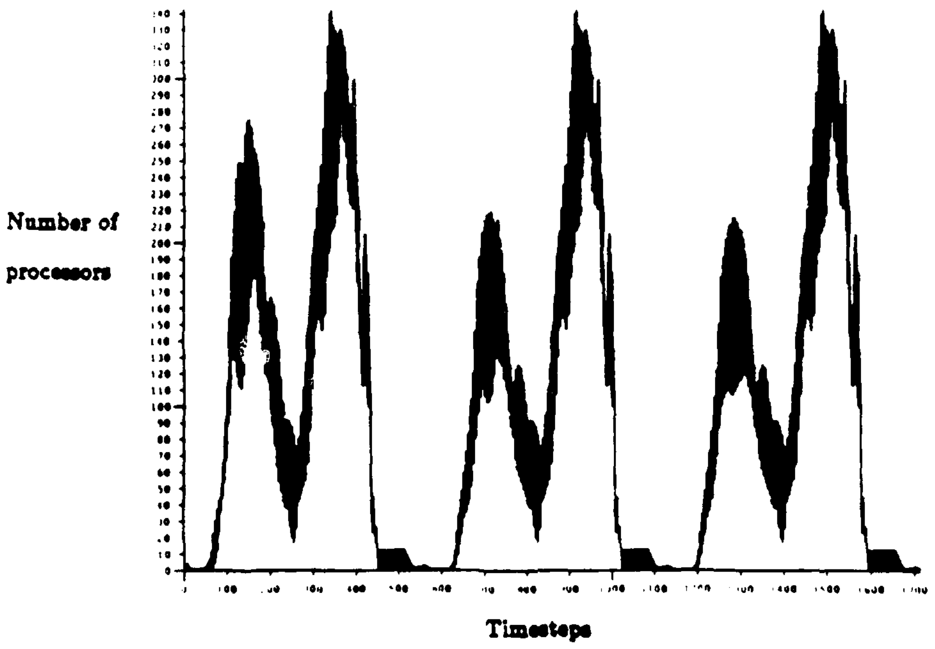


Figure 7: Floating Point Operations Profile with Loop Bounding

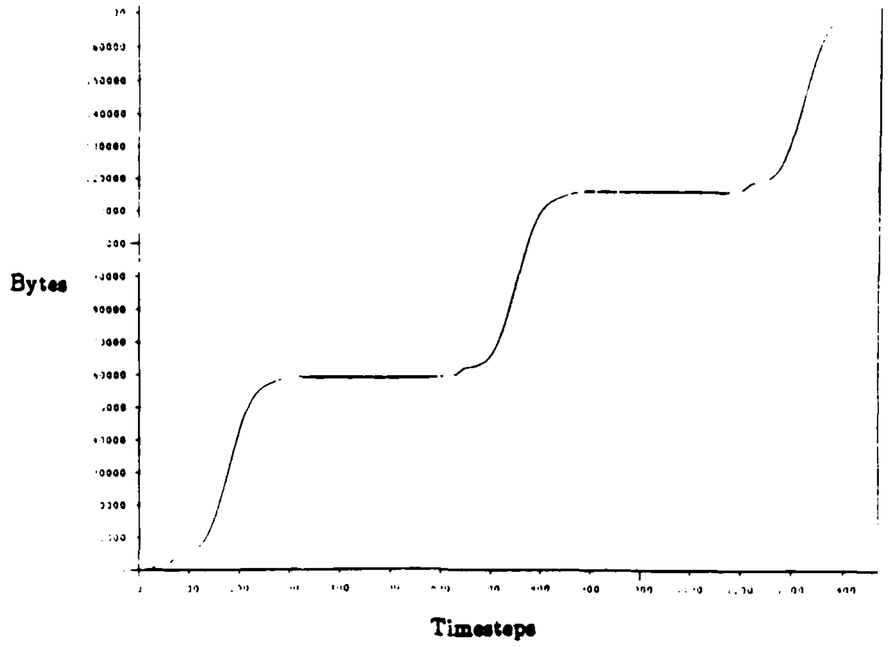


Figure 8: I-Structure Storage Profile with Loop Bounding

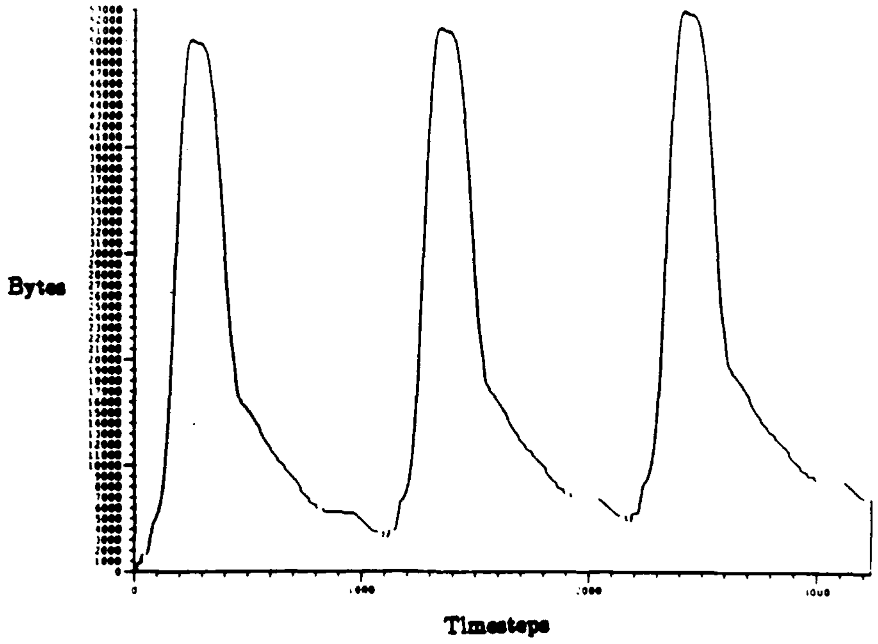


Figure 9: I-Structure Storage Profile with Loop Bounding and Release

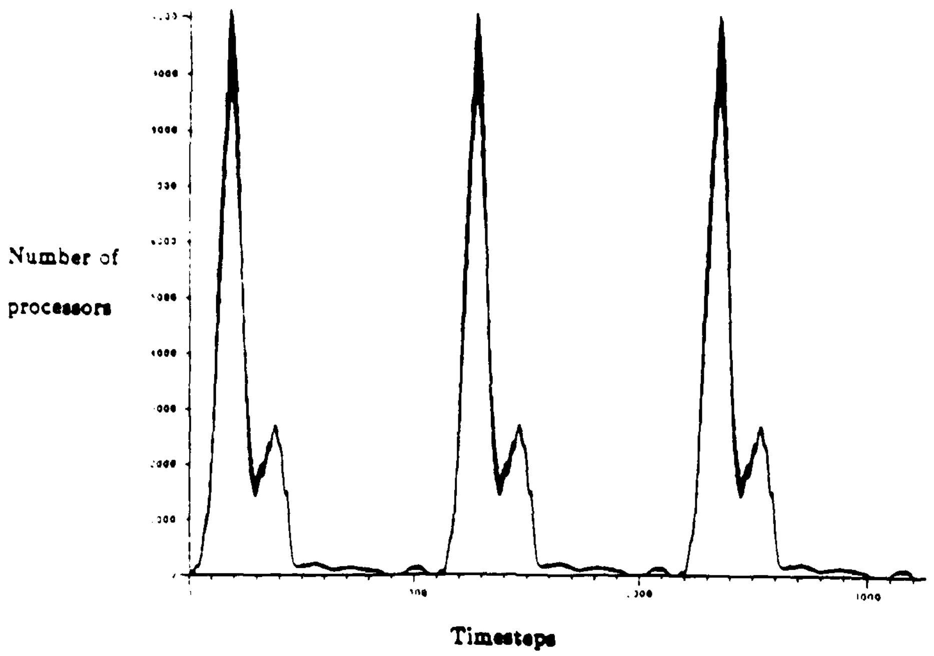


Figure 10: ALU Operations Profile with Loop Bounding and Release

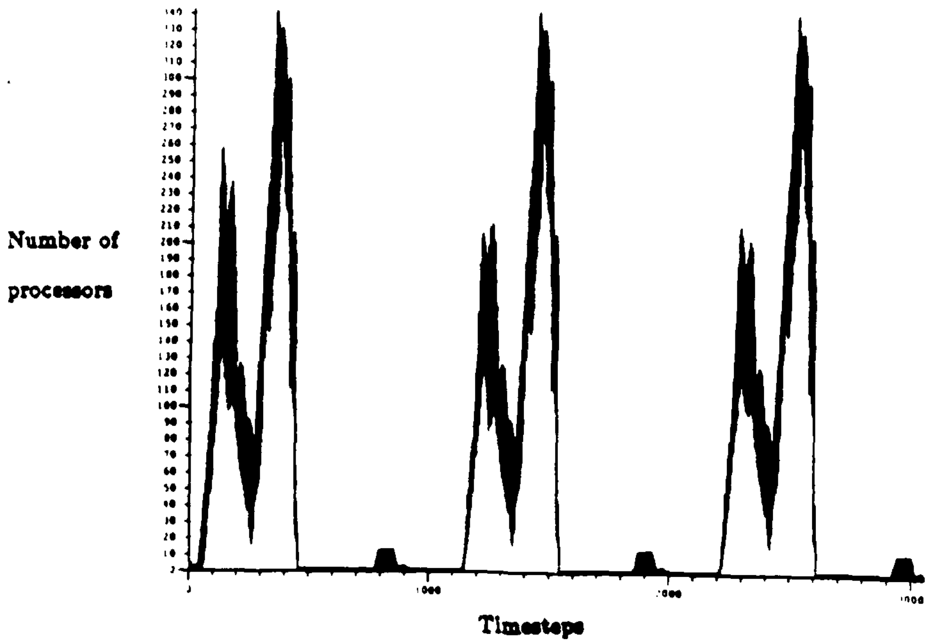


Figure 11: Floating Point Operations Profile with Loop Bounding and Release

Adaptation Methods for Viscous Flows

John G. Kallinderis
Judson R. Baron

Dept. of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, MA

Invited Chapter

in

Computational Methods in Viscous Aerodynamics

Editor C.A. Brebbia

Springer-Verlag

Adaptation Methods for Viscous Flows

John G. Kallinderis and Judson R. Baron

Computational Fluid Dynamics Laboratory, Dept. of Aeronautics and Astronautics Massachusetts Institute of Technology, Cambridge, MA 02139

INTRODUCTION

Considerable progress has been made in recent years in the development of numerical methods for the simulation of viscous flows. Various approximations to the Navier-Stokes equations have been used and different numerical schemes have been developed for the prediction of viscous fields. The traditional approach solves a specific set of equations on a single grid using the same numerical scheme over the entire domain. The selection of the equations, the scheme and the grid are determined *a priori* by the user before starting the solution procedure. However, quite often some or all of the above factors must be modified by the user in order to improve the results.

The robustness of current numerical schemes as well as present computer capabilities has allowed a dramatic change in this philosophy. General algorithms have been developed which are flexible enough to adaptively adjust the equations, the grid and even the scheme during the solution procedure without intervention by the user. In other words, the algorithm makes decisions with respect to which equations, grids or schemes are to be used as the computation proceeds.

First we discuss several adaptation methods as well as other concepts which can be applied for the efficient simulation of fluid flows, emphasizing those which are viscous. The reasoning that leads to developing adaptive algorithms is examined, and the methods of grid, equation, and temporal adaptation are described. Next, the problem of detecting the flow features of interest, the communication between different grids and the application of a turbulence model are discussed. Lastly, the special coding requirements for an adaptive algorithm, example problems, as well as an evaluation of the methods in terms of accuracy and efficiency are presented.

NEED FOR ADAPTIVE ALGORITHMS

Most of the current schemes (especially those used for complex transonic flow problems) are of low order accuracy. Higher order methods often require a large computational molecule which complicates the algorithm and poses a serious problem when applying boundary conditions. Numerical schemes of low order of accuracy are usually used when shocks are present in order to capture them without oscillations. The use of a higher order scheme also may result in extra computation and storage of additional quantities other than state variables (e.g. stresses). Imposing high spatial accuracy often leads as well to severe restrictions on the size of the time-step. Lastly, the order of smoothing operators that are currently used degenerates when non-smooth grids are used. Therefore, in practice schemes are at most second order accurate in space and time. However, the accuracy is grid dependent and schemes often become first order or even inconsistent if the grid is stretched, skewed, etc. As a consequence increased resolution is needed in the vicinity of features to ensure accurate predictions. A boundary layer, for example, requires at least 15 points across its thickness and the point adjacent to the wall may lie at a distance of order 10^{-6} for a high Reynolds number flow. The resolution constraint, orthogonality and other grid quality requirements combine to make grid generation very difficult. The problem becomes more severe in 3-D where the generation of body-fitted, nearly orthogonal grids is very difficult.

The initial approach to alleviate the grid problem has been a modification of the initial grid during the computation by redistribution of the grid points such that more points are clustered in the regions of interest [18,17,7]. However, the number of grid points is fixed and clustering in one region results in less resolution in other regions. Another drawback of the method is that it frequently results in skewed and stretched grids which deteriorate the accuracy of the integration scheme.

Another method, which will be described in some detail here, is the use of a coarse grid initially into which are embedded finer grids in those local regions with large flow gradients (e.g. boundary layers, shocks, wakes, etc). Several levels of finer grids can be used, and can be limited to those regions of the domain where important features exist. A special feature detection algorithm can be introduced in order to place the grids adaptively during the course of the solution procedure.

Within a flow domain the dominant physics differs from region to region since the flow often exhibits a variety of flow phenomena with different characteristics,

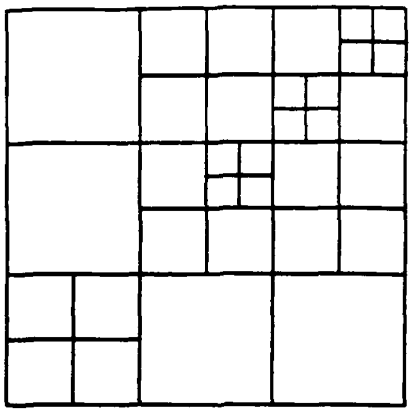


Figure 1: Grid embedding

such as shocks and shear layers. These regions can be described by different sets of equations. For example, the Navier-Stokes equations may be necessary in a boundary layer but the Euler equations actually suffice to describe the physics for an inviscid part of the flow. An expensive Navier-Stokes calculation need not be carried out over the entire domain. An adaptive algorithm which can sense the presence of specific physical phenomena, and which uses the appropriate governing equations or solver, seems to be advantageous in terms of CPU time savings.

It is clear that an algorithm which adapts itself to the developing solution by means of modifying the grid, the equations and/or perhaps the scheme itself, can offer flexibility and economy.

ADAPTIVE LOCAL GRID REFINEMENT

The objective of adaptive grid refinement is to adjust the grid scale in regions where extra resolution is needed (Fig. 1).

The resulting embedded grids are topologically similar to the initial grid and so maintain its geometric properties (e.g. stretching, orthogonality), but are not necessarily aligned to the initial grid as the embedded meshes 'follow' the features. The process can be repeated any number of times and results in finer and finer local embedded grids until a region is adequately resolved. This approach has been employed for the Euler equations [6,4,12,15], and for Navier-Stokes [10,11,9]. However, the existence within the same domain of features with specific but different orientations has implications for the way in which an initial grid is embedded. Divi-

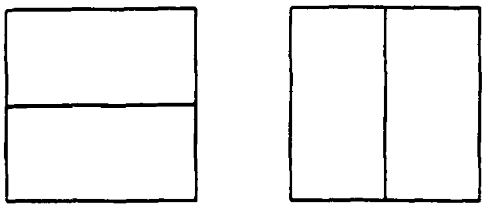


Figure 2: Directional embedding

sion of a cell in both directions creates three additional cells and frequently results in unnecessary resolution in regions where significant flow gradients exist primarily in a single direction. For example, in a boundary layer extra resolution usually is needed in the crossflow direction, but not streamwise. The advantages offered by directional cell division (Fig. 2) are therefore clear: only one additional cell is created instead of three, and this results in significant savings in computation time and storage [10].

Local embedding implies two important consequences for any basic solver that is used. First, the mesh now becomes unstructured and the usual i,j indexing can no longer be used. Instead, the solver sweeps through cells and the necessary operations are restricted to within each cell. A system is required to keep track of all the needed information for each cell (pointer system), and will be described below. Second, there is an implied communication between the grids. The borders between grids of different refinement levels (interfaces) must receive special attention.

EQUATION ADAPTATION

The Navier-Stokes equations apply for most flow fields of engineering interest. Frequently, not all of its terms are needed to model the flow physics. The viscous terms in fact are expensive to compute but often are negligible over large parts of the domain. A suggested approach introduces a criterion based on the magnitude of the viscous stresses in order to allow the algorithm to decide where the full Navier-Stokes system is required and where a subset system (e.g. the Euler equations) would be adequate. The border between two such regions is dynamically defined by the algorithm and may change during the course of the solution procedure [10]. Figure 3 illustrates the concept for a shock/boundary layer interaction problem.

This procedure essentially decouples the solver from the grid. Different integrators can be used in different regions of the domain and splitting of the domain offers flexibility in defining an integration strategy. Not all of the regions need be

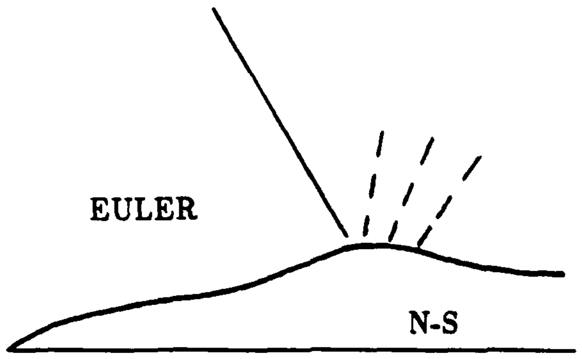


Figure 3: Adaptive splitting of the domain into viscous, inviscid regions

visited during each sweep. A shear layer developes much more slowly during the solution procedure due to the very small time steps used there, while the outer inviscid flow advances rapidly to a steady state. Especially in cases without any strong viscous/inviscid interactions, a strategy can be adopted which integrates the viscous regions a number of times for each inviscid region integration.

Such viscous/inviscid decomposition of the domain does not create additional interface problems since the same solver is used but with the negligible viscous terms omitted in the inviscid region. However, interfaces may exist for cases in which different schemes are employed in each of the regions. The adaptive domain decomposition approach may be extended to more than two regions. For example, Euler equations may be applied in the inviscid region(s), the full Navier-Stokes in viscous regions where significant streamwise diffusion exists, and the thin layer Navier-Stokes in those boundary layer regions with negligible streamwise diffusion.

TEMPORAL ADAPTATION

For time-accurate explicit methods, the size of the time step is restricted by numerical stability considerations such as the CFL condition. There are situations, however, in which the time step constraint is governed by physics rather than stability considerations. This can occur when there are significant time gradients which must be resolved, and then a procedure similar to that for spatial embedding may be used. The algorithm monitors the time-gradients within each cell and, when above a specified threshold, the time step is reduced. Both time and spatial interfaces appear and generally differ from one another [13].

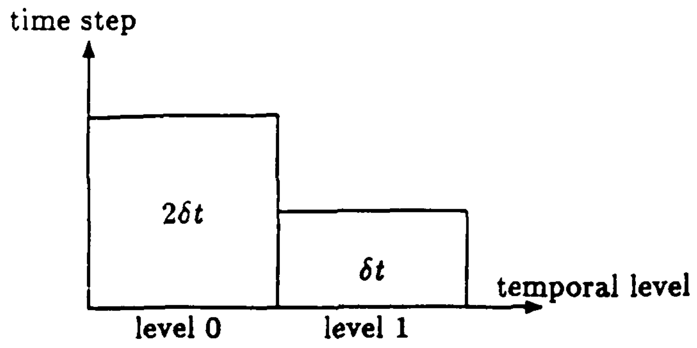


Figure 4: Time steps in adjacent embedded zones differ by a factor of two

In order to simplify the procedure, the time steps that are allocated to each cell may be grouped into temporal zones. All cells within a temporal zone then are integrated with the same time step and the time steps between temporal zones vary by factors of two as illustrated in Figure 4, and the maximum time-step in the domain is 2^n times larger than the minimum one, where n is the number of embedding zones. The cells of temporal level 1 are integrated in time twice using a time step δt before the cells of temporal level 0 are integrated once using a time step of $2\delta t$.

This procedure allows a spatial variation of the time steps while simultaneously maintaining time accuracy. A further simplification insists that temporal interfaces coincide with the spatial interfaces. This can be accomplished by taking into account the fact that spatial adaptation (cell subdivision to resolve spatial gradients) often results in temporal adaptation (reduction of the time step to resolve temporal gradients) as well. Large temporal gradients may exist in regions where features are present. Figure 5 illustrates the concept with the temporal zones coinciding with the spatial ones. This splitting of the time-steps according to the embedded zones saves CPU time since not all of the cells are marched in time with the minimum time-step that is found for the entire domain. Instead, only those cells that are in the embedding zone containing the cell with the minimum time-step are integrated with that minimum time-step; the remaining cells which lie on other zones, are marched at time-steps that are multiples of the globally smallest time-step.

ADAPTIVE BLOCKS

Two major constraints are imposed on the integration scheme by these procedures. The first is that explicit solvers should be used with an unstructured mesh. The presence of interfaces in the domain makes the formulation of an implicit scheme

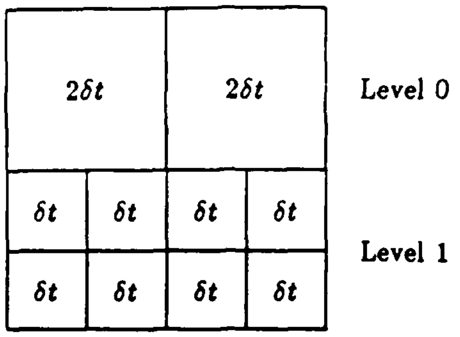


Figure 5: Spatially varying time steps

quite difficult. The second constraint restricts all operations to using information from within each cell to upgrade properties for that cell. Most current schemes, such as that in the preceding finite volume chapter, satisfy these constraints and do not seem to impose a serious limitation in the use of adaptation procedures.

However, there are applications for which an implicit scheme may be more advantageous. In such a case, the adaptive grid philosophy can still be used by defining structured groups of cells (blocks). These can be formed by completely embedding portions of the domain rather than embedding individual cells. A way of constructing such blocks is by using the 'embedding patches' concept which is described in the feature detection section. Within each block an implicit solver may be used. The block approach can apply to only a specific region of the domain (e.g. a boundary layer, which can be considered as a single block), while the totally unstructured grid can be employed for the rest of the field.

ADAPTATION STRATEGIES

Consider how the described adaptation methods can be combined into a single procedure during a computation. A typical sequence of steps for the steady-state algorithm is:

1. Solve the Navier-Stokes equations on an initial coarse grid.
2. Monitor the residual error until it falls below a prespecified value; detect the main flow features and the borders between the viscous and the inviscid regions; refine the grid locally if the detection parameter exceeds the threshold for division, or remove grids if it is less than the threshold for removing grids.
3. Continue computing on the updated grid using the Navier-Stokes equations only within the viscous region and Euler everywhere else.

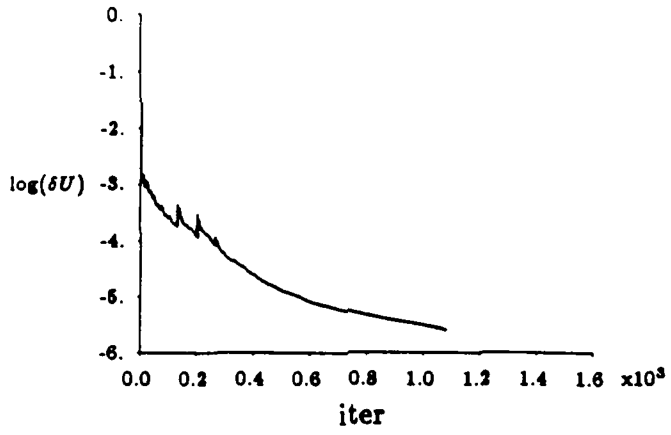


Figure 6: Convergence history with three levels of adaptation

4. Repeat steps 2,3 for a specified number of adaptation cycles.
5. March solution to steady state.

Figure 6 shows the rms residual decay to a steady-state during which three adaptations were completed. The residual peaks after embedding due to interpolation error when allocating values of the state variables to the newly created nodes. The jump magnitude becomes less as the grids become finer.

Of course, the above adaptation strategy is not unique. It can and should be modified to make it more suitable for specific areas of application. For example, if interest is in forces acting on an airfoil, the embedding process logically can be related to the grid convergence for lift and drag. In that case the solution should be marched to a provisional steady-state for lift, for example, after each adaptation, instead of tracking the residual decay.

Additional techniques may be included depending upon the nature of the special features. One such technique involves the use of preembedding. The initial grid is embedded in a certain local region before the computation starts in order to meet a specific need. For example, in a shock tube problem an accurate prescribing of the initial shock is accomplished with embedding. Similarly, when a vortex needs to be prescribed in a region as an initial condition, the initial coarse mesh may be inadequate to prescribe it accurately and therefore the vortex region may be pre-embedded. After initiating the computation the embedding may no longer be needed or be required elsewhere due to the feature's motion. The algorithm then automatically removes the pre-embedding.

FEATURE DETECTION

The detection of individual features and of different regions is an essential part of all adaptation algorithms. The adaptation algorithm must be able to sense the existence and track the evolution of such features.

There are two basic approaches in detecting features. The first is based on truncation error estimates [2,5]. Since a primary objective of adaptation is minimization of this error, it is evident that direct detection of those regions for which the error is large is a most suitable guide. Unfortunately, the truncation error is not known directly. The order of the numerical scheme that is used in every region is required, and the solution on two different grids must be examined along with the scheme's order of accuracy, in order to estimate the truncation error. The order of accuracy of the scheme, however, is not known exactly since it is usually grid dependent and is not the same over the entire domain. In a shock region, for example, a second order scheme usually reduces to first order. Another drawback of the truncation error method is the high cost of error computations. This is undesirable especially for unsteady evaluations in which the detection process is applied frequently.

The basis for the second approach is detection of regions where significant flow variations exist [6,10]. Of course, the truncation error depends upon field variations. It is negligible in the regions where such derivatives are small and maximum in regions of larger flow variations. Therefore, it is reasonable to track regions of large flow gradients and to resolve them, since they are generally the sites at which flow features exist. Detection of these regions essentially leads to detection of the flow features. Viscous flow fields exhibit a great variety of features with different character, length scales, and orientation. A shock ideally has zero thickness while a boundary layer basically has a finite thickness. Also, shocks can have different orientations, shear layers very often follow surface boundaries, while a wake, on the other hand, frequently takes the direction of the mean flow. Both orientation as well as the location of the features may change appreciably in the course of a computation. In a steady-state computation, the gradual thickening of a boundary layer may cause the shock to move appreciably, while oscillations of the free stream may change the orientation of a wake considerably.

Feature detection is grid dependent. A very coarse grid 'sees' even very mild flow variations while a fine one may 'ignore' large changes. This is the driving force of both spatial and temporal adaptation. The grid is adjusted so that it 'sees' a relatively smooth solution which means low truncation error. The choice of appropriate feature detection parameters is guided by the physical nature of the flow. Across a boundary layer the viscous stresses clearly are important and informative, while the approximately constant pressure is irrelevant. On the other hand, pressure gradients are good indicators of a shock region. Entropy and Mach number variations are not good indicators of weak shocks. Conversely, density differences are good indicators of both shocks and boundary layers, but clearly inappropriate for incompressible flow. Depending upon the kind of problem numerous other detection parameters may be employed. Temperature, for example, may be quite effective in detecting the overheated regions in the first stage of a turbine and may result in good resolution of regions where accurate heat transfer prediction is important. Another class of parameters that may be applied are those which use information from the field geometry. For example, if complete resolution of the trailing edge region of an airfoil is desired, distance from the trailing edge can be used as a criterion for adapting that area, and a detection parameter ΔU may be modified to $\Delta U.R_{TE}$, with R_{TE} being the distance fron the trailing edge. It is quite clear that no one universal parameter applies for all applications, and that more than one parameter may be useful tc achieve accurate feature detection.

The form of the detection parameter(s) is another choice that must be made in carrying out the feature detection procedure. There are two main forms: undivided and divided (gradients) differences, and each may be of first or higher order. An undivided difference can be the difference in the detection parameter between the opposite corners (NE, SE and NW, SW) of a cell

$$\delta U = (U_{NE} + U_{SE}) - (U_{NW} + U_{SW}).$$

A divided difference is formed by normalizing the above difference by a characteristic cell-dimension e.g.

$$\frac{\delta U}{\delta X} = \frac{(U_{NE} + U_{SE}) - (U_{NW} + U_{SW})}{(X_{NE} + X_{SE}) - (X_{NW} + X_{SW})}.$$

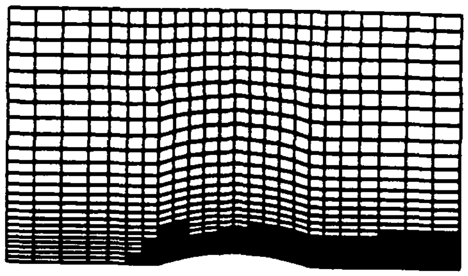
Again the choice of the form is problem dependent. For example, first order gradients (stresses) provide good resolution for the inner part of a boundary layer but leave the outer edge poorly adapted. Second order gradients are needed to resolve the latter. The cells within a viscous layer are generally smaller than those in the inviscid regions. As a consequence, an undivided difference of a detection parameter can be of the same magnitude for both inviscid and viscous cells. Therefore, divided differences are appropriate for proper capture of shear layers. On the other hand, use of parameter gradients when detecting shock regions leads to increasing gradients after each adaptation, and most probably an overadaptation of the shock region and an ignoring of the rest of the flow domain. Thus, a combination of both divided differences for shear layers and undivided differences for shocks is appropriate. For example, if $(\frac{\delta U}{\delta X})_{cell} \geq (\frac{\delta U}{\delta X})_{th}$ or $(\delta U)_{cell} \geq (\delta U)_{th}$ where $(\frac{\delta U}{\delta X})_{th}$ and $(\delta U)_{th}$ refer to threshold values, then the cell is divided in the X-direction. Still another consideration in the choice of detection parameters is the amount of computational effort, which may be important when the detection process is repeated frequently. In figures 7, 8 are shown the performance of various criteria in the detection of a boundary layer and a shock separately in a channel flow with a bump. Fig. 7 illustrates a subsonic flow field with a boundary layer being the main feature, while Fig. 8 shows the same field in supersonic flow. In the latter case a shock is formed at the leading edge and is reflected at the upper boundary. It evident that the use of density leads to excessive number of embedded cells. As was mentioned, Mach number is a poor indicator of weak shocks compared to pressure and velocity differences which perform quite well.

The feature detection process requires specific threshold values for each detection parameter. One can use statistics for the determination by calculating the average ($\overline{\Phi} = \frac{\Sigma_{cells}\Phi_{cell}}{N}$) and standard deviation ($\sigma_{\phi} = \frac{\sqrt{\Sigma_{cells}\Phi^2_{cell}}}{N}$) of the values of the detection parameter Φ over all N cells in the domain. The threshold then is defined as the average parameter value plus a fraction (α) of the standard deviation $(\Phi)_{th} = \overline{\phi} + \alpha\sigma_{\phi}$ (see Fig. 9).

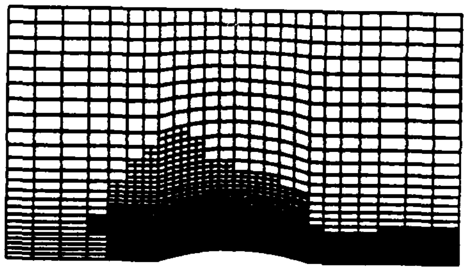
The value of the coefficient α is chosen empirically, and a typical value is 0.4. The detection process is not critically dependent upon the choice of α.

After selecting those cells to be adapted, a smoothing procedure can be applied to the grid. The procedure aims at eliminating spurious cells that may appear which are essentially 'noise cells', or cells which should have been detected but were overlooked by the procedure. Figure 10 shows characteristic situations in which the grid has both 'holes' and 'islands'. Simple rules can be constructed so as to eliminate these situations.

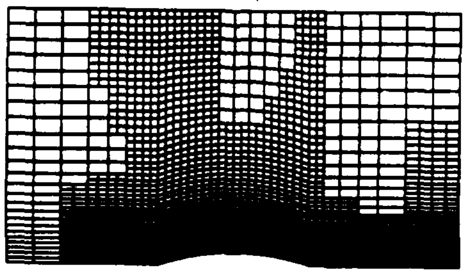
Another way of smoothing the grid is the use of 'embedded patches' [11] . An embedded patch encloses a defined, fixed number of cells of the initial mesh. It essentially scans the domain and during the scan the included cells are examined for each patch. If the majority of its cells (e.g. 90%) are flagged for division, then all cells currently belonging to the patch are flagged for division. Conversely, if very few cells (e.g. 10%) are flagged, none within the patch are embedded. When



(a) velocity gradients ($\frac{\delta_l u}{\delta l}, \frac{\delta_m u}{\delta m}, \frac{\delta_l v}{\delta l}, \frac{\delta_m v}{\delta m}$)

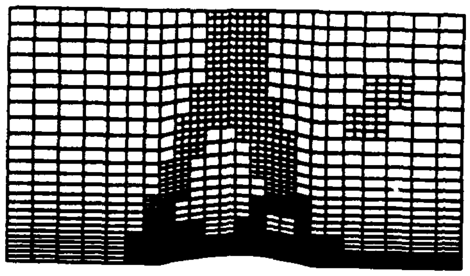


(b) velocity differences ($\delta_l u, \delta_m u, \delta_l v, \delta_m v$)

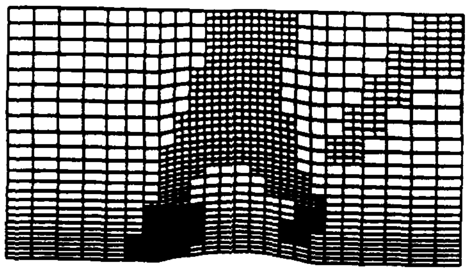


(c) density differences ($\delta_l \rho, \delta_m \rho$)

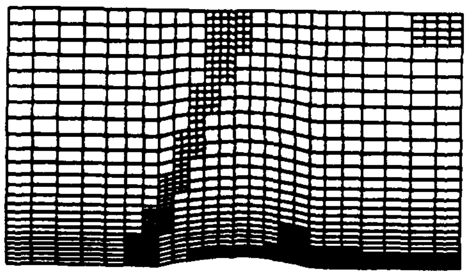
Figure 7: Detection parameter influence on adaptation for subsonic boundary layer



(a) velocity differences ($\delta_l u, \delta_m u, \delta_l v, \delta_m v$)



(b) pressure differences ($\delta_l p, \delta_m p$)



(c) Mach no differences ($\delta_l M, \delta_m M$)

Figure 8: Detection parameter influence on adaptation for a shock

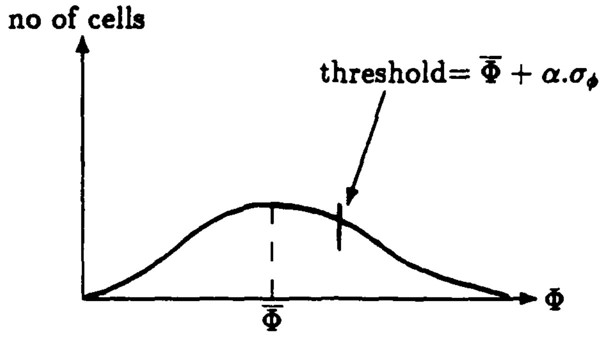


Figure 9: Determination of threshold value for embedding

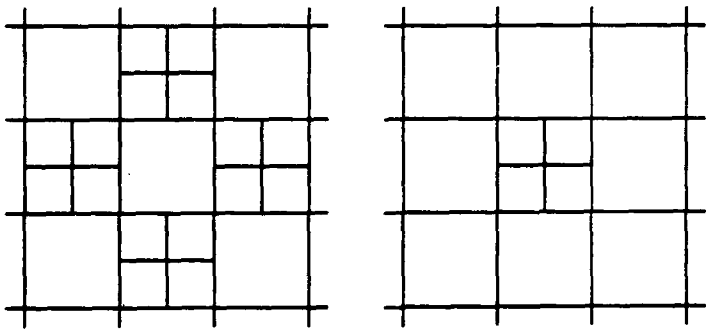


Figure 10: Unacceptable grid formulations

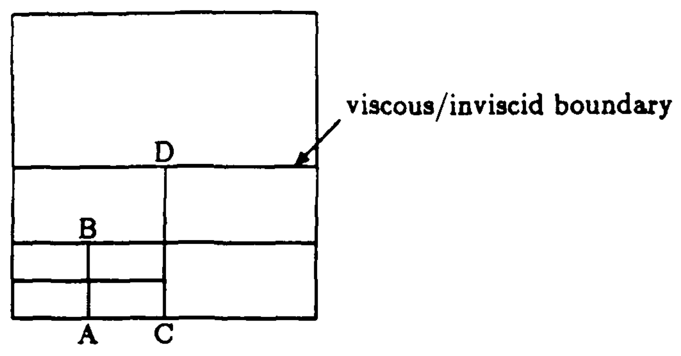


Figure 11: Interruption of normal lines by grid interfaces

repeated throughout the domain, the patch procedure can be viewed conceptually as a 'searching window', and acts somewhat like a 'noise filter' which reduces the number of 'randomly' embedded cells.

AN ALGEBRAIC TURBULENCE MODEL WITH UNSTRUCTURED GRID

The implementation of an algebraic turbulence model (e.g. Baldwin-Lomax or Cebeci-Smith) in the presence of unstructured grids introduces difficulties. The models implicitly assume a structured mesh, and the implementation is usually along lines perpendicular to the surface. With an unstructured mesh continuous normal mesh lines do not necessarily exist because of interrupting interfaces (Fig. 11), and clearly the models cannot be directly applied along lines such as AB and CD. This may be overcome by applying the models in a 'cell-wise' manner, based on parameters known at the center of each cell. In this way we avoid using information from outside the cell, an approach which is a common procedure when dealing with embedded meshes generally [10,9].

Consider now in more detail how the Baldwin-Lomax algebraic turbulence model [3] might be applied to an unstructured mesh. The cells in the turbulent region are arranged in streamwise stations (Fig. 12) and consist of those from the initial mesh plus those introduced by a the first embedding level. The first embedding level defines the borders between viscous and inviscid regions and therefore completely covers the turbulent region. During the higher than first level refinements no additional streamwise stations are created.

Two kinds of quantities are necessary: those that characterize each cell and others that characterize each station. Vorticity and the distance of the center of the cell to the wall are both cell-based. The wall quantities y^+, u^+ and the Baldwin-Lomax parameters $Y_{max}, F_{max}, U_{diff}$ characterize the entire station. The vorticity is

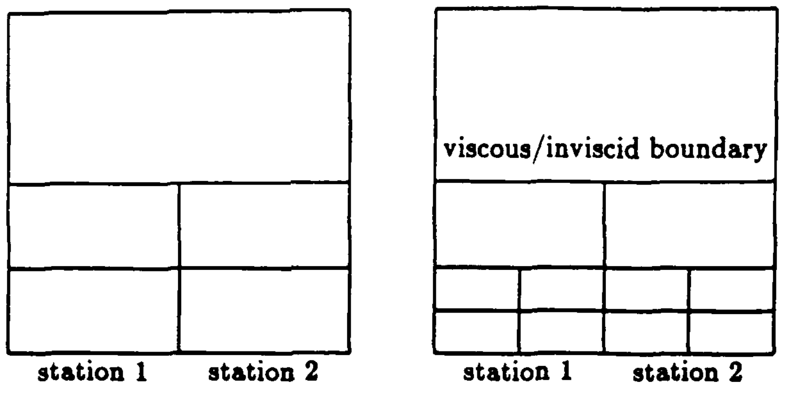


Figure 12: Cells grouped by stations

calculated using Green's theorem over each cell: $\omega = -\frac{1}{S_{cell}} \oint_{cell} (udx + vdy)$ where S_{cell} is the cell area. The distance of each cell from the wall is calculated and stored whenever the grid is updated. The station quantities Y_{max}, F_{max} are calculated by scanning through all of the cells that belong to each station. The function $F(y) = y|\omega|(1 - \exp\frac{-y^+}{A^+})$ is formed for each cell and its maximum value F_{max} which occurs at distance Y_{max} from the wall is found. The wall quantities y^+, u^+ are evaluated as averages from the station cells which are adjacent to the wall. The basic assumption is that there is no significant variation in F_{max}, Y_{max} over the streamwise length of a station.

The treatment for wakes is similar to the above with few modifications. The wake stations are arranged in pairs (Fig. 13) formed from upper and lower parts. The minimum velocity cell is found by scanning through the cells of both stations of each pair. Then cells 'migrate' from one station to its counterpart so that those which are above the minimum velocity cell are assigned to the upper station, and the remaining are assigned to the lower station of the pair. The normal distances of the cells from the center of the wake then are modified by simply adding or subtracting the normal distance over which the wake moved from the stored normal distances for each cell.

A verification of the above approach has been carried out for the case of a NACA 0012 airfoil in a flow of $Re = 2.91 \times 10^6$, $M_\infty = 0.5, \alpha = 1.77°$. An initial C-mesh of 33x17 points and two levels of embedding were used for the computation [11]. Fig. 14(a) shows the pressure coefficient C_p distribution for the laminar case, while in Fig. 14(b) the turbulent case is compared to experiment. The computed C_l 0.197 may be compared with the experimental value of 0.195.

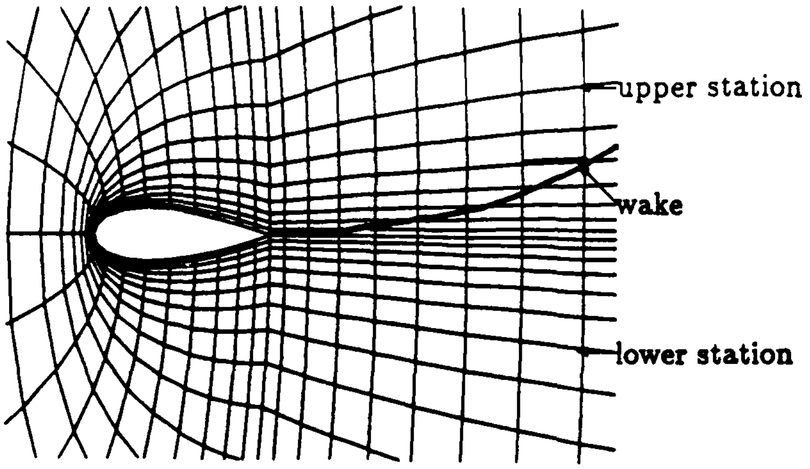
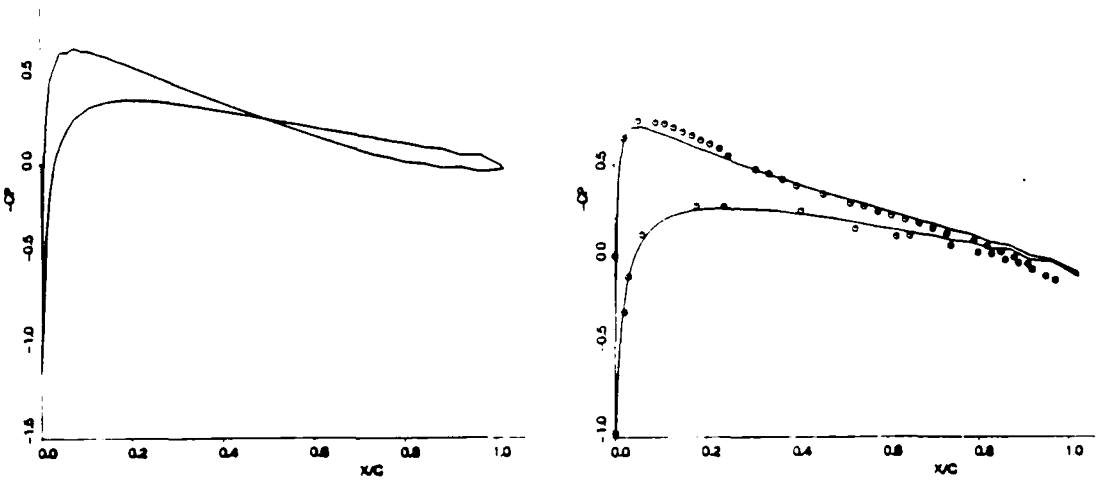


Figure 13: Allocation of cells to turbulent wake stations



(a) laminar

(b) turbulent
- present work [11]($Cl = 0.197$)
o experiment [16] ($Cl = 0.195$)

$M_\infty = 0.5, Re = 2.91 \times 10^6$

Figure 14: Pressure coeff. distribution for NACA 0012 airfoil

CODING WITH UNSTRUCTURED MESHES

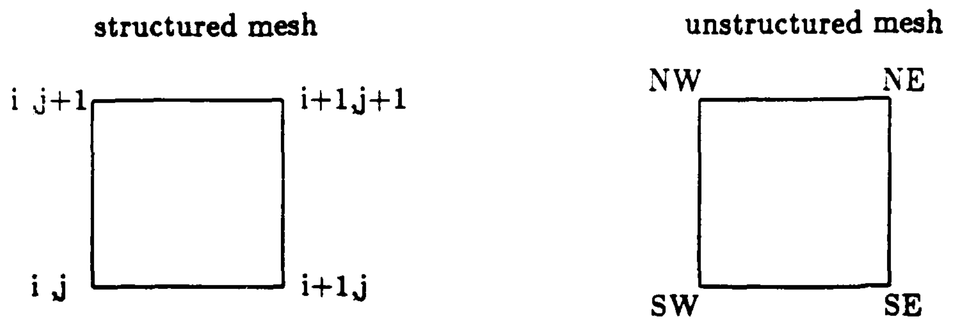
A classical structured grid is composed of quadrilateral cells which are arranged so that each grid coordinate has an (i,j) index. State variables are then defined at particular points in a two-dimensional array, the essential point being that neighbouring points in the array structure are also neighbouring points in the physical computational domain. Such an arrangement no longer exists when unstructured grids are used, in which case all of the necessary operations are generally performed in a 'cell-wise' manner. The domain consists of labeled cells, and is scanned over cells rather than (i,j)-nodes. All cells are stored in a one-dimensional array and use is made of indirect addressing to obtain the values of the variables and other information that may be needed from the corners of each cell.

It is informative to consider a single Do-loop example for an unstructured mesh and the corresponding loop for a structured mesh. Say that the loop computes the value of u-velocity at the south face of a cell (Fig. 15). For the unstructured mesh the four corners are stored in the IC array. The double Do-loop in the structured mesh case is replaced by a single loop of twice the length in the case of the unstructured mesh. This is a desirable feature since vectorization is generally more efficient for lengthy Do-loops. On the other hand, more memory is needed for the pointers used for the indirect addressing. There is also an extra computational cost associated with gather/scatter operations related to the addressing. However, memory overhead is becoming less important as storage capabilities increase, and the cost of the gather/scatter operations is reducing as computer architectures are optimized for the operations which are used by the unstructured codes.

COMMUNICATION BETWEEN CONTIGUOUS GRIDS (Interfaces)

The existence of embedded regions within the interior of the computational domain introduces internal boundaries (interfaces), which may be either spatial or temporal or both. For example, in Fig. 5 there are both spatial and temporal interfaces which coincide.

An interface is generally characterized by an abrupt change in the cell and time-step size which poses problems of accuracy. The grid lines may either continue across the interface [8] or they may be interrupted by the interface [10,14]. In the latter case, cells appear which contain additional nodes at the faces. Existing schemes have been developed for cells with nodes at only the four corners and they need



```
   DO 10 J= NJ
   DO 10 I= NI
10 US= 0.5*{ U(I,J) + U(I+1,J) }
```

```
   DO 10 CELL= 1,NCELL
   SW= IC (1,CELL)
   SE= IC (4,CELL)
10 US= 0.5*{ U(SW) + U(SE) }
```

Figure 15: Differences in coding with structured and unstructured grids

some modification in order to take into account the extra face nodes.

A number of considerations must be taken into account when designing a numerical scheme involving interfaces. Perhaps the foremost concern is maintaining accuracy at an interface despite the stretching error that is related to the sudden change in grid size. Second-order numerical schemes degrade to first-order and in some cases become inconsistent where the grid is stretched. Special care is required for face (hanging) nodes which are not surrounded by four cells. Another important issue is the maintenance of conservation, by which is meant that interface fluxes between cells cancel one another at a common interface ($F_1 + F_2 + F_3 = 0$ in Fig. 16).

Most common smoothing operators are of second order for shock capturing and fourth order for the suppression of the spurious oscillations that may appear. Both operators have a stencil larger than a single cell, which means that they gather information from both sides of the interfaces. Unfortunately, interface stretching deteriorates the smoothing operator accuracy. An interface treatment scheme should also be 'free-stream preserving', meaning that it accepts uniform flow as a solution. Other important considerations are the coding complexity, and the ease with which the interface treatment scheme can be extended to three dimensions. In practice, the number of interfaces in a domain of $O(N^2)$ points is only $O(N)$. As a consequence,

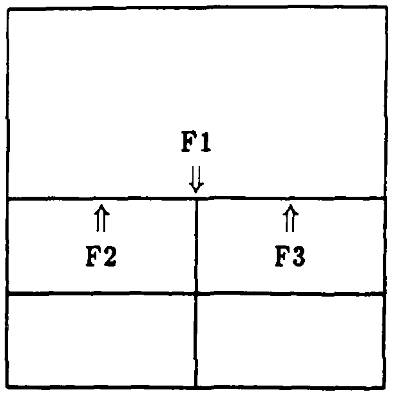


Figure 16: Conservative treatment: interface fluxes should cancel

the added computing time due to an interface is not an issue.

It is clear that the above considerations impose serious limitations on the construction of an interface scheme, and that in many cases these can be contradicting requirements. The simultaneous achievement of both conservation and accuracy is very difficult, and even impossible to achieve in most of the cases. However, not all of the above factors are important for a specific interface. For regions in which gradients are small, the lack of accuracy and/or conservation has a negligible effect on the numerical results. Conservation proves to be an important property in cases of moving shocks for the accurate prediction of their location and speed. However, it is not important inside a shear layer. Conversely, accuracy may be more of an issue in a boundary layer since the second order derivatives (viscous terms) are important and they are more 'sensitive' to the grid stretching error than first order derivatives, which are important in the inviscid regions.

Let us now describe a non-conservative and a conservative interface treatment. The non-conservative treatment is illustrated in Fig. 17(a). The nodes a,b are integrated using the parent cell B instead of the embedded cells C and D, while ignoring the face node c. The values at node c are obtained by interpolating the values from nodes a,b. The same approach can be employed for all kinds of interfaces (e.g. Fig. 17b). This treatment is also easily extended to three dimensions. Although it is non-conservative it does avoid the stretching error associated with the grid discontinuity at the interface. The nonconservation error is of $O(\Delta x)$ and it is local in the interface region only. Therefore, in most of the cases it does not deteriorate the accuracy of the solution appreciably. Only in cases of a shock that is located very close and parallel to the interface or when a shock moves through an interface, an appreciable error is expected. A more detailed investigation of the various interface

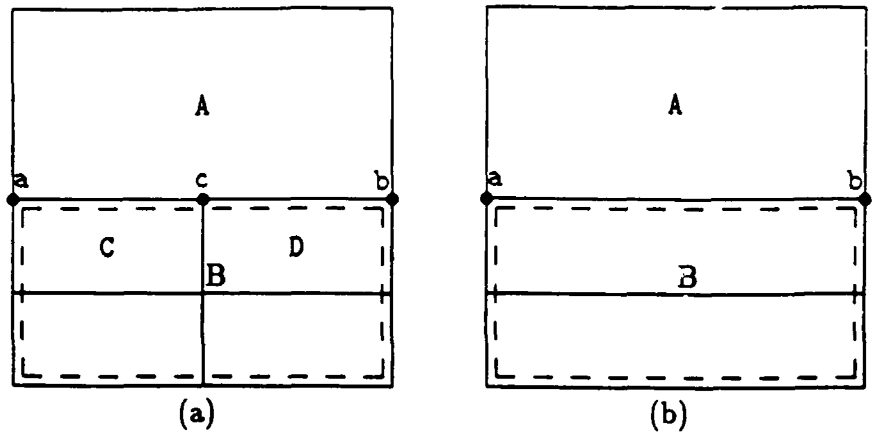


Figure 17: Nonconservative interface treatment

treatments and especially the issue of nonconservative vs conservative treatments, can be found in [9].

The treatment can be made conservative by performing a special integration in cell A that takes into account the face node c. For a trapezoidal integration of the flux of a quantity U around cell A the interface fluxes are:

$$\begin{aligned} F_A = 0.5(U_a + U_c)(X_c - X_a) &\;+\; 0.5(U_c + U_b)(X_b - X_c) \\ -0.5(U_a + U_c)(Y_c - Y_a) &\;-\; 0.5(U_c + U_b)(Y_b - Y_c). \end{aligned}$$

This flux is cancelled by the corresponding fluxes from cells C and D

$$F_C = 0.5(U_c + U_a)(X_a - X_c) - 0.5(U_c + U_a)(Y_a - Y_c)$$

$$F_D = 0.5(U_b + U_c)(X_c - X_b) - 0.5(U_b + U_c)(Y_c - Y_b).$$

In the normal case of a cell with four nodes, the cell-area is divided into four equal areas, one for each node and therefore a fourth of the total cell-change is allocated to each node. However, when a fifth face node exists, the cell area is allocated to the five nodes as shown in Fig. 18(b) with the resulting distribution coefficients shown in Fig. 18(a). A similar 'geometric' approach can be applied to obtain the distribution for the cases of cells with two, three and four face nodes. A more rigorous derivation of the distribution formulas, which uses shape functions has been employed and verifies the above 'geometric' approach [1]. However, this treatment suffers from the stretching error which is especially severe when the viscous terms are important. For example, cell *ABCD* (Fig. 18(c)) is used to evaluate u_{xx} at node *b* which is not at the cell-center.

For time accurate computations, where time and spatial-interfaces coincide, both cell-size and time-step size change together (Fig. 5). The non-conservative treatment

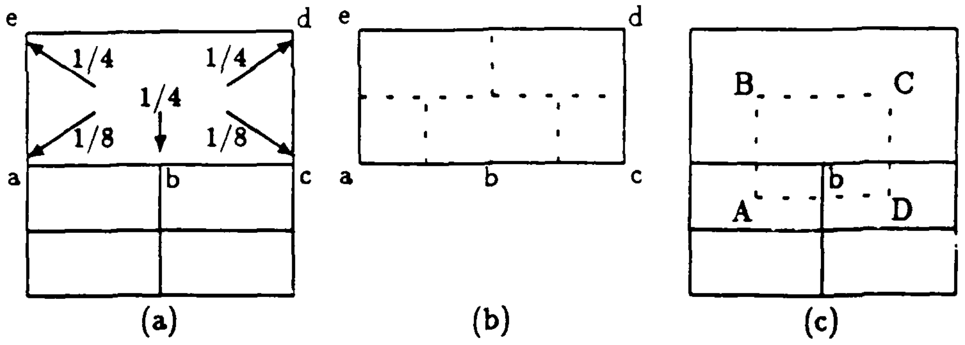


Figure 18: Conservative interface treatment

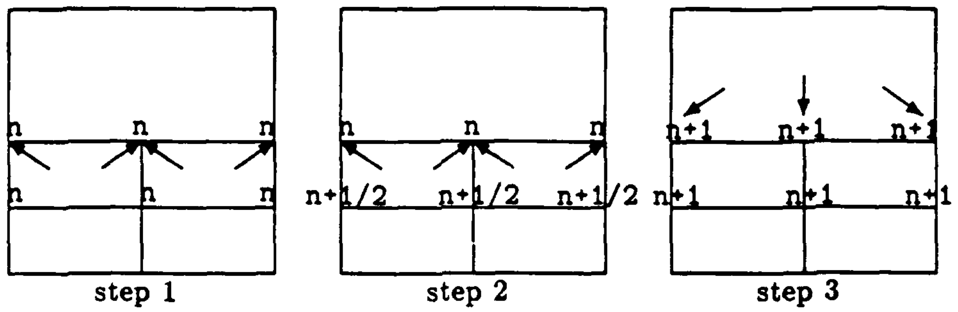


Figure 19: Conservative interface treatment in time

remains basically the same. The interface nodes a,b and c are considered a part of the coarser level and are integrated once with a time-step of $2\delta t$, after the finer cells are integrated twice with time-step of δt. However, special care must be taken in order for the interface fluxes to cancel each other and make the treatment conservative (Fig. 19). During the first and second steps the fine cells are integrated at time level n but the distributions to the interface nodes as shown by arrows in the figure are saved for later updating. At the third step, the coarser cells are integrated and now the saved distributions from the fine cells are assigned together with those from the coarser cells. Since the interface nodes are kept to the old time level n during the first two steps, the interface fluxes cancel each other at the third step and the treatment becomes conservative.

EXAMPLES OF ADAPTIVE CALCULATIONS

Some examples of results will best illustrate the capabilities of adaptive algorithms. In all the cases the non-conservative interface treatment was used.

The first example involves supersonic flow through an 8% circular arc cascade, with an entrance $M_\infty = 1.4$ and $Re = 23 \times 10^3$ [10].

An initial mesh of 25x25 was used followed by three levels of embedding. An oblique shock forms at the leading edge of the arc section and it is reflected at the upper symmetry boundary. The reflected shock then interacts with the boundary layer at the trailing edge region. In Fig. 20, the grid evolution demonstrates how the embedded grids follow the detailed physics of the flow. Figure 20(e) provides an enlarged detail of the grid near the surface and Figure 21 shows the flow field in terms of Mach number and C_p contour plots. The boundary layer is essentially 'lifted' by the adverse pressure gradient which in turn is induced by the reflected shock. Simultaneously, because of the effective corner which is formed by the boundary layer, compression waves are formed upstream of the interaction region and coalesce into a weaker shock which impinges on the upper boundary. Note that passage of the shock through interfaces does not induce any stability problems. The local embedding procedure appears to effectively capture the detailed physics of a flow field in the presence of rather complicated multiple-scale phenomena. Equation adaptation was also used to limit the solution of the Navier-Stokes equations to only within the viscous region.

The feature detection procedure and in particular the utility of the embedding patches together with directional embedding are evaluated in a second example. The case of a NACA 0012 airfoil in a flow of $Re = 10^5$ and $M_\infty = 0.8$ and an angle of attack of $2°$, shows the effectiveness of the feature detecting procedure [11]. Figure 22(a) shows Mach number contours immediately prior to embedding (note the enlarged vertical scale). The dominant features that are apparent are a normal shock on the suction side and two boundary layers, one on either side of the airfoil. Figure 22(b) shows the resulting adaptive grid. It can be seen that there is directional embedding above and below the airfoil, while the leading edge region is embedded in both directions. The downstream region between the two shear layers remains unembedded. Comparison of the grid and the resulting solution makes quite evident that the features are faithfully captured. The borders between the different regions are fairly free from 'noise' cells. Figure 23 shows an enlarged view of the flow field (Mach contours) for the separation region on the suction side of the airfoil. The flow details appear to be very well captured by the method. In cases completed so far, there appears to be no restriction on the number of adaptations carried out during a computation.

ADAPTATION EVALUATION

The previously described adaptation methods must also be considered in terms of accuracy and CPU time savings.

Accuracy

The essential advantage of spatial embedding is that resolution is introduced only into regions where large flow gradients exist. Since regions with relatively uniform flow do not need resolution, an embedded grid provides an accuracy equivalent to a globally fine grid. All of those errors which are introduced due to interfaces (e.g. non-conservation and stretching errors) scale with the size of the interface cells. As the algorithm refines the region where a feature exists, interfaces are introduced in gradually finer regions and all errors diminish. The outer (first) level interfaces are susceptible to the largest errors but are generally positioned away from the largest gradients regions and do not introduce appreciable inaccuracies. Typically, for a boundary layer the first level interface is normally placed outside of the layer's edge, while finer embedded grids appear within the layer. Apart from the interface issue, the accuracy of the embedded grids is normally very good since the refinement is governed by the flow gradients: larger gradients imply finer embedded grids. Figure 24 compares skin friction distributions for the previously described supersonic circular arc cascade but for two-level embedding and a globally fine grid of 97x97 [10]. The agreement is excellent. Figure 25 shows a separated velocity profile for the same arc cascade in subsonic flow (M_∞), and the presence of three interfaces does not introduce any kinks to the profile. The use of equation adaptation does not introduce accuracy questions since the viscous terms are omitted only where they are negligible. Their neglect is a very accurate approximation.

The same remarks that where made for the spatial embedding are applicable to temporal embedding as well. Errors related to temporal accuracy scale with time-step size which means that they are proportional t~ ~ell size. Figure 26 illustrates the case of a flow with an oscillatory Mach nu~ the inlet ($M_\infty = 0.8 + 0.04 sint$), and for which there are significant temporal gradients over the entire domain [11]. One level of embedding has been used as shown in the figure. Both curves represent time histories of the U-velocity component at a specific node of the domain. The agreement is very good.

Two other adaptation properties improve the numerical accuracy. The first is a diminishing of the smoothing error which usually scales with the cell size. This is most important within the viscous regions where smoothing can contaminate the solution seriously. The approach that often has been used switches off smoothing gradually as the wall is approached. This allows considerable smoothing in regions with large gradients which are away from the wall (e.g. shock/boundary layer interaction region). Embedding inherently provides a way of diminishing this error and quite often switches off smoothing gradually as more viscous regions are approached wherever they may be. The second is the ability to make use of a good initial mesh (minimal stretching, orthogonal). The embedded grids have the same good properties as the initial mesh.

A question does arise with respect to the reliability of the feature detection. Accuracy depends to a great extent upon the correct detection of the flow features. The two main factors of uncertainty that exist in the detection procedure are the choice of appropriate detection parameters and the determination of accurate threshold values. Experience to date indicates that it is relatively easy to decide what parameters should be used for certain classes of problems and that the threshold value choice is not so crucial. A worst case may provide excessive embedding for some feature, which translates into less CPU savings; however, this will not affect the accuracy.

CPU time savings

The original motivation for the introduction and use of adaptation methods is a reduction in CPU time without loss of accuracy. The savings in computation time have proven to be quite significant in most cases.

Several main factors govern the amount of CPU time savings which an embedded grid offers as compared to a structured grid of the same resolution level. First is the number and extent of the flow features with respect to the size of the computational domain. The greater the extent, the lesser the savings; the larger the number of embedding levels, the higher the savings. A large number of embedding levels allows the use of a relatively coarse initial mesh, and any comparison with a globally fine mesh with the same refinement level as the finest embedded grid indicates high CPU savings. However, there are some limitations on how coarse the initial mesh can be. In a high Reynolds number case, the final spacing at the wall may be of the order of 10^{-6} which means that the initial mesh will have a wall spacing of the order of 10^{-5} which is still quite fine. The initial mesh should have some points inside the boundary layer (approximately four). Otherwise, the Navier-Stokes solver will 'see' a sudden jump in velocity from zero to free-stream and it may diverge. Also, a very coarse initial mesh may yield inaccurate feature detection which will 'guide' adaptation in the wrong way. This is corrected at the next adaptation, but some CPU time is wasted.

Let us now proceed to an examination of the efficiency of each one of the adaptation methods. Grid embedding is the most effective and offers the most savings. The savings are proportional to the ratio R of the cells of the equivalent globally fine mesh to those of the embedded mesh. Each time an initial mesh with N cells is embedded, the resulting number of cells is $N + 3fN$ where f is the fraction of initial cells which are divided. Therefore, after k adaptations, the ratio R is:

$$R = \frac{4^k}{(1+3f_1)(1+3f_2)...(1+3f_k)}.$$

N does not enter the above expression directly, however, it affects the number of adaptations k, since the use of a relatively coarse initial mesh leads to a large number of embedding levels k in order to reach a certain refinement level. The value of the embedded cell fraction f, depends upon the extent of the features to be resolved in the domain. If features cover the most of the field, then $f \to 1$ and thus, $R \to 1$, which implies that the saving is not important. Conversely, if the features are confined to a small portion of the domain, $f \to 0$ and $R \to 4^k$. Also, the larger the number of adaptations k, the bigger is R. Typical values for f may be $f_1 = 0.3, f_2 = 0.15, f_3 = 0.1, f_4 = 0.05$, which yields $R \approx 62$. Directional embedding leads to significant savings by virtue of introducing only one extra cell compared to the three additional when division in both cell-directions is applied. In this case, the above ratio R is:

$$R = \frac{4^k}{(1+f_1)(1+f_2)...(1+f_k)}.$$

It should be emphasized that the savings in CPU-time due to embedding is not governed only by the above ratio R. The solution in an embedded mesh advances to steady-state much faster than in the corresponding globally fine mesh since there are coarse regions in the domain which allow considerably larger time-steps. Equation adaptation is less effective and in fact, did not provide significant savings for viscous two dimensional cases since the majority of the cells are inside the boundary layer. Nevertheless, significant savings are to be expected when appreciable resolution is needed in the inviscid region and in three dimensional applications. The use of time-steps that vary spatially according to the embedding zones for time accurate calculations leads to significant savings provided that the sizes of the cells within

	CPU time savings factor
Globally Fine	1
Adaptation(nondirectional)	13
Adaptation(directional)	24
Equation Adaptation	28

Table 1: CPU time savings due to adaptation

the same embedding zone are similar. If the cell-sizes vary significantly within the same embedding zone, then significant savings are not expected since all cells are integrated with the time-step of the smallest cell of the embedding zone. Table 1 shows the CPU time savings factor for each of the methods when applied to the steady state field generated by a circular arc cascade in subsonic flow ($M_\infty = 0.5$). Embedding in both directions gives a speed-up factor of 13, directional embedding increases the factor to 24, and equation adaptation to 28, which is the total CPU time savings factor due to the combination of all adaptation methods. For the unsteady case of an oscillating inlet Mach number flow, the use of spatially varying time-steps provides 25% CPU time savings per period with the use of only two embedding levels, compared to the identical embedded mesh using the globally minimum time-step. Finally, updating pointers after each adaptation does not increase the computation time appreciably, especially if adaptation is applied only a few times during the computation.

CONCLUDING REMARKS

Viscous flows are very demanding in terms of computation time and grid quality. The current low order numerical schemes on conventional structured meshes are quite expensive. Algorithms which can modify both the grid and the equations used in the course of a computation in order to resolve flow physics, seem to be promising for more efficient viscous computations. Methods such as *adaptive lo-*

cal grid refinement, *equation adaptation* and *temporal adaptation* have been applied quite successfully yielding significant savings in CPU-time without sacrificing accuracy. On the other hand, the use of unstructured (embedded) grids introduces extra complications. A data structure system is needed in order to keep the required information, and interfaces require special treatment. Also, the detection of features is very important for accuracy. These tasks, however, can be successfully tackled in many cases and do not seem to impose serious problems in the application of adaptive algorithms to flows of engineering interest.

Acknowledgement This work was supported by AFOSR Grant 82-0136 , Drs J.D. Wilson, J. McMichael, and L. Sakell serving as technical monitors.

References

[1] S.R. Allmaras. *private communication.*

[2] I. Babuska(Ed.). *Accuracy Estimates and Adaptive Refinements in Finite Element Computations.* J. Wiley and Sons, 1986.

[3] B. S. Baldwin and H. Lomax. *Thin-Layer Approximation and Algebraic Model for Separated Turbulent Flows.* AIAA Paper 78-257, 1978.

[4] M. Berger and A. Jameson. Adaptive grid refinement for the euler equations. *AIAA Journal*, 23:561–568, April 1985.

[5] M. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comp. Phys.*, 53:484–512, 1984.

[6] J. F. Dannenhoffer III and J. R. Baron. *Grid Adaptation for the 2-D Euler Equations.* AIAA Paper 85-0484, 1985.

[7] P.R. Eiseman. Grid generation for fluid mechanics computations. *Ann. Rev. of Fluid Mechanics*, 17:, 1985.

[8] K.A. Hessenius and T.H. Pulliam. *A Zonal Approach to Solution of the Euler Equations.* AIAA Paper 82-0969, 1982.

[9] J. G. Kallinderis. *Space, Time and Equation Adaptation for Viscous Flows.* Ph.D. Thesis, Massachusetts Institute of Technology, to appear Spring 1989.

[10] J. G. Kallinderis and J. R. Baron. *Adaptation Methods for a New Navier-Stokes Algorithm.* AIAA Paper 87-1167-CP, 1987(to appear in AIAA Journal Sept. 1988).

[11] J. G. Kallinderis and J. R. Baron. *Unsteady and Turbulent Flow using Adaptation Methods.* Proceedings of the 11th Int. Conf. on Numerical Methods in Fluid Dynamics, Williamsburg, Virginia, 1988.

[12] R. Lohner, K. Morgan, J. Peraire, and O. C. Zienkiewicz. *Finite Element Methods for High Speed Flows.* AIAA Paper 85-1531, 1985.

[13] M.M. Pervaiz and J. R. Baron. Temporal and spatial adaptive algorithm for reacting flows. *Comm. in Applied Numerical Methods*, 4(1):97–111, January 1988.

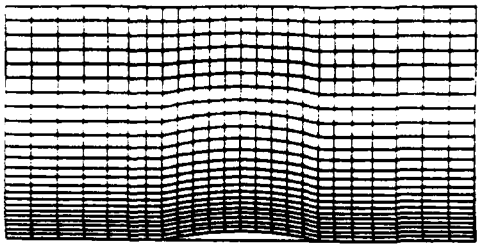
[14] M.M. Rai. *A Relaxation Approach to Patched-Grid Calculations with the Euler Equations.* AIAA Paper 88-0034, 1988.

[15] R. A. Shapiro and E. M. Murman. *Adaptive Finite Element Methods for the Euler Equations.* AIAA Paper 88-0034, 1988.

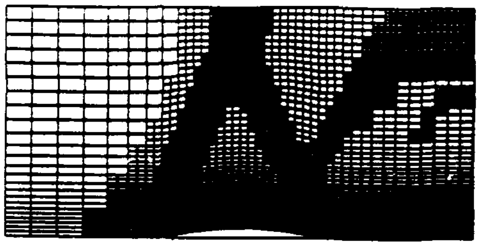
[16] J. J. Thibert, M. Granjacques, and L. H. Ohman. *NACA 0012 Airfoil.* Technical Report AGARD AR 138, AGARD Advisory Re ort, 1979.

[17] J.F. Thompson. *Review on the State of the Art of Adaptive Grids.* AIAA Paper 84-1606, 1984.

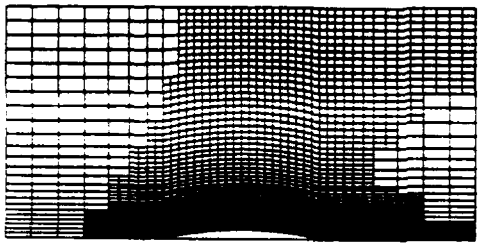
[18] J.F. Thompson, Z. Warsi, and C.W. Mastin. *Numerical Grid Generation.* Elsevier Science Publishing Co., Inc., 1985.



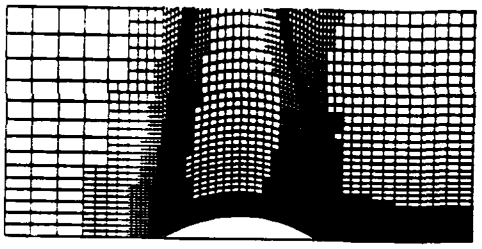
(a) initial grid (25x25)



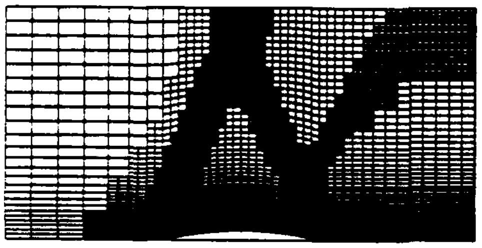
(d) third embedding level



(b) first embedding level



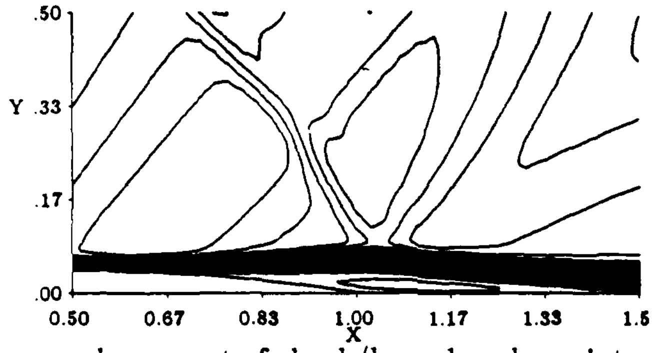
(e) third embedding level
(vertical scale enlarged)



(c) second embedding level

Figure 20: Grid evolution for 8% circular arc cascade ($M_\infty = 1.4$)

Mach contours (increment= 0.04)



enlargement of shock/boundary layer interaction region

Pressure coeff. contours (increment= 0.03)

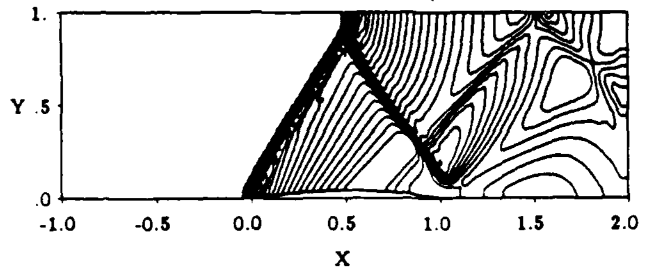
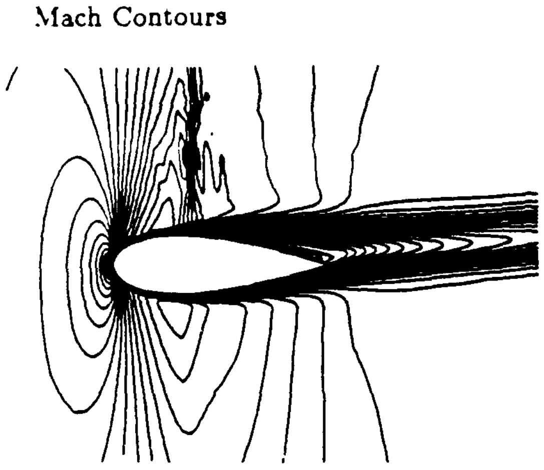
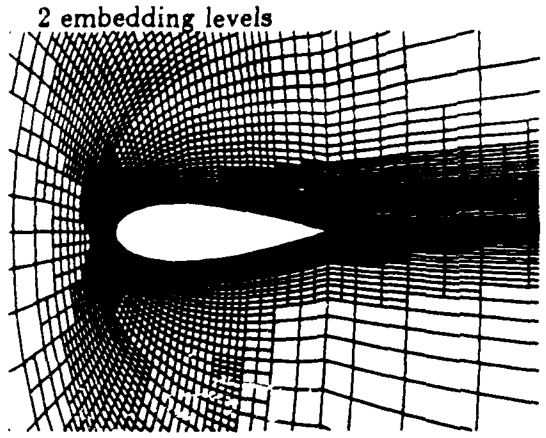


Figure 21: Solution contours for 8% circular arc cascade ($M_\infty = 1.4$)



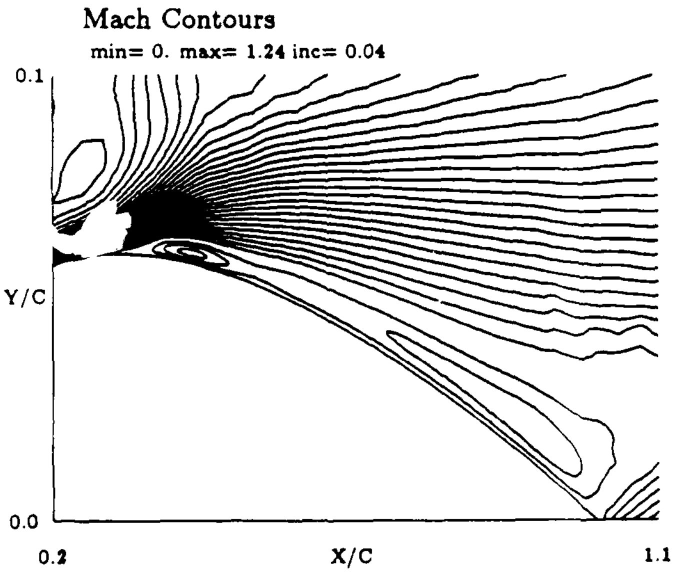
(a) solution before embedding



(b) grid with Embedding Patches

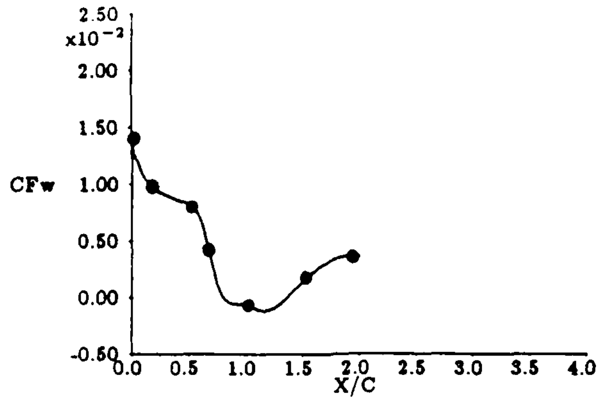
NACA 0012 ($M_\infty = 0.8, Re = 10^5, \alpha = 2^\circ$), vertical scale enlarged

Figure 22: Effectiveness of embedding patches in feature detection



NACA 0012 $M_\infty = 0.8, Re = 10^5, \alpha = 2^\circ$

Figure 23: Enlargement of separated region



– embedded grid (2-levels)

● globally fine grid (97x97)

Figure 24: Comparison of skin friction distributions for adapted, globally fine grids

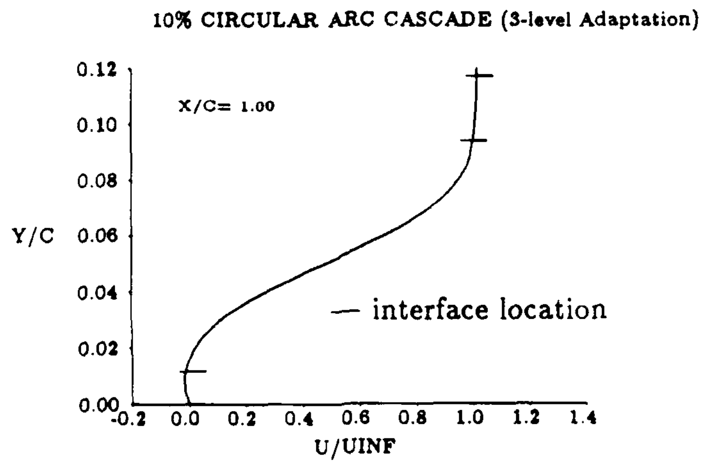


Figure 25: Separated boundary layer profile in the presence of interfaces

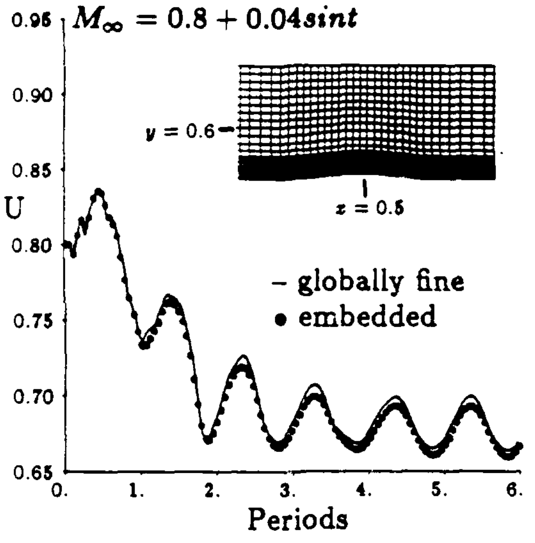


Figure 26: Velocity history at $x = 0.5$, $y = 0.6$ for oscillating inlet flow

APPENDIX 14

The Finite Volume Approach for the Navier-Stokes Equations

John G. Kallinderis
Judson R. Baron

Dept. of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, MA

Invited Chapter

in

Computational Methods in Viscous Aerodynamics

Editor C.A. Brebbia

Springer-Verlag

The Finite Volume Approach for the Navier-Stokes Equations

John G. Kallinderis and Judson R. Baron

Computational Fluid Dynamics Laboratory, Dept. of Aeronautics and Astronautics Massachusetts Institute of Technology, Cambridge, MA 02139

INTRODUCTION

The Navier-Stokes equations have a range of validity that covers most of the flows of engineering interest. Their numerical solution is expensive and quite often a subset of the equation system is employed. However, numerous problems require the use of the full Navier-Stokes equations for either the description of the entire flow field, or the description of limited regions of the domain. Rapid progress in computer technology has made the solution of the Navier-Stokes equations less expensive and, although it cannot yet be ranked along with the standard theoretical tools currently used in applied aerodynamics, it is certain that in the near future it will be used in routine engineering calculations.

Methods that have been used for the numerical solution of the Navier-Stokes equations can be divided into four main categories: (i) finite difference , (ii) finite volume , (iii) finite element , and (iv) spectral methods. Finite difference and finite volume methods are the most widely used at present.

We will consider the finite-volume approach in general without going into specific descriptions of the various schemes that have been developed. Surveys of numerical schemes for solving the Navier-Stokes equations may be found in [12,7,8]. The emphasis here will be on the basic principles and on those problems that are common to most of the schemes. These include such issues as accuracy, viscous grid requirements, smoothing, etc. First, the Navier-Stokes equations are presented and the finite-volume discretization is described. Next, finite-volume approaches to evaluating viscous terms are examined, and issues related to spatial accuracy and requirements of a viscous grid are addressed, including the effects of artificial dissipation on accuracy especially within the shear layer. Lastly, a conservative, finite-volume scheme that has recently been developed will be presented and investigated with respect to the issues that have been addressed.

NAVIER-STOKES EQUATIONS

The system may be written in cartesian two-dimensional conservation form as :

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = \frac{\partial R}{\partial x} + \frac{\partial S}{\partial y}$$

where

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ E \end{pmatrix}, F = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ (E+p)u \end{pmatrix}, G = \begin{pmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ (E+p)v \end{pmatrix}$$

are state and convective flux vectors in the x and y- directions respectively. The viscous flux vectors are

$$R = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ u\tau_{xx} + v\tau_{xy} - q_x \end{pmatrix}, S = \begin{pmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ u\tau_{xy} + v\tau_{yy} - q_y \end{pmatrix}$$

where $\tau_{xx}, \tau_{yy}, \tau_{xy}$ are viscous stresses, and q_x, q_y are heat conduction terms. They are given by the following relations:

$$\tau_{xx} = (\lambda + 2\mu)\frac{\partial u}{\partial x} + \lambda\frac{\partial v}{\partial y}$$

$$\tau_{yy} = (\lambda + 2\mu)\frac{\partial v}{\partial y} + \lambda\frac{\partial u}{\partial x}$$

$$\tau_{xy} = \mu(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x})$$

$$q_x = -\kappa\frac{\partial T}{\partial x}, \ q_y = -\kappa\frac{\partial T}{\partial y}$$

In the above relations ρ is density, u and v are velocity components, E is total internal energy per unit volume, p is pressure and T is the temperature. For a perfect gas, the pressure is related to the specific total internal energy E by

$$p = (\gamma - 1)\left[E - \frac{\rho}{2}\left(u^2 + v^2\right)\right].$$

Also, λ and μ are the viscosity coefficients and κ is the coefficient of thermal conductivity. The Stokes relation is used to eliminate the bulk viscosity coefficient λ

$$3\lambda + 2\mu = 0$$

and μ is a known function of temperature T (e.g. Sutherland's law).

FINITE VOLUME DISCRETIZATION

The above two-dimensional Navier-Stokes equations are considered in the integral form

$$\int\int_S \frac{\partial U}{\partial t} dS + \int\int_S (\frac{\partial F}{\partial x} + \frac{\partial G}{\partial y}) dS = \int\int_S (\frac{\partial R}{\partial x} + \frac{\partial S}{\partial y}) dS \tag{1}$$

The surface integrals are evaluated by using Green's theorem, which is the heart of the finite-volume approach. Consider an area S and its contour ∂S which is simply connected and piecewise smooth, a vector $\vec{V} = (P, Q)$ that is twice continuously differentiable, and the unit vector $\vec{n}$ which is normal to the contour ∂S.

One form of Green's theorem in cartesian coordinates is:

$$\int\int_S (\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}) dxdy = \oint_{\partial S} \vec{V}.\vec{n} dl = \oint_{\partial S} (Pdx + Qdy). \tag{2}$$

The calculation of a surface integral is thus reduced to the evaluation of a line integral. Using Eq.(2) the integral form of the Navier-Stokes equations (1) becomes:

$$\frac{\partial}{\partial t} \int\int_S U dS + \oint_{\partial S} (Fdy - Gdx) = \oint_{\partial S} (Rdy - Sdx). \tag{3}$$

The flow domain S consists of smaller areas (cells). The cells are defined through the coordinates of their vertices which is the only grid information that is needed. The integrals over the entire domain S in Eq.(3) may be replaced by the sum of integrals over each cell:

$$\sum_{cells} \frac{\partial}{\partial t} \int\int_{cell\ area} U dS + \sum_{cells} \oint_{cell\ faces} (Fdy - Gdx) = \sum_{cells} \oint_{cell\ faces} (Rdy - Sdx). \tag{4}$$

There is a physical interpretation for each of the above terms. The first term expresses the total change with time of a quantity (e.g. density) for all cells, while the second term is the sum of the net fluxes of the same quantity through the faces of the cells. The term on the right hand side represents viscous fluxes through the faces.

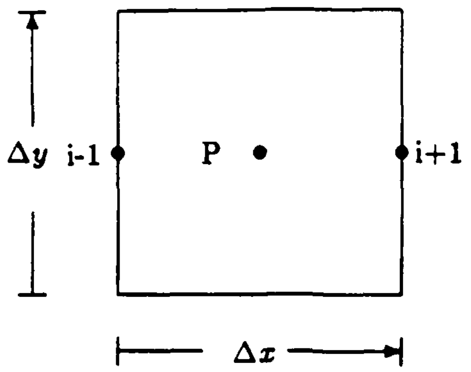


Figure 1: Equivalence of finite difference and volume modeling

If the flow field is uniform, then equation (2) yields [6,15]:

$$\oint_{\partial S} \vec{n} dl = 0 \tag{5}$$

which means that the cells should be closed surfaces. A second geometric constraint is that the sum of the cell-areas should equal the area of the entire domain. Both geometric constraints together basically 'define' the kind of control areas that may be used, which are usually quadrilaterals and triangles.

Consider the relation between the finite-volume and finite-difference approaches taking as an example an evaluation of the first order derivative $\frac{\partial U}{\partial x}$ at the point P in the cartesian cell of (Fig. 1). Using the finite-volume approach,

$$\iint_S \frac{\partial U}{\partial x} dS = \oint_{\partial S} U dy \Rightarrow (\frac{\partial U}{\partial x})_P = \frac{1}{S}\oint U dy = \frac{1}{\Delta x \Delta y}(U_{i+1}\Delta y - U_{i-1}\Delta y) = \frac{U_{i+1} - U_{i-1}}{\Delta x}$$

which is identical to central finite-differencing.

Green's formula provides a derivative value which is averaged over the cell-area but does not necessarily apply at the center of the cell. In fact, the substitution $(\frac{\partial U}{\partial x})_P = \frac{1}{S}\oint u dy$ holds only if P is located at the centroid of the cell. This can be seen from a Taylor series expansion of $\frac{\partial u}{\partial x}(x, y)$ around the point (x_0, y_0) so that

$$\iint_S \frac{\partial U}{\partial x} dS = \iint_S \frac{\partial u}{\partial x}(x_0, y_0) dS + \iint_S (x - x_0)\frac{\partial^2 u}{\partial x^2}(x_0, y_0) dS + \iint_S (y - y_0)\frac{\partial^2 u}{\partial y^2}(x_0, y_0) dS + ..$$

Therefore,

$$\frac{\partial u}{\partial x}(x_0, y_0) = \frac{1}{S}\int\int_S \frac{\partial U}{\partial x} dS$$

holds exactly only if

$$\int\int (x - x_0)^m (y - y_0)^n dS = 0$$

for $m, n = 0, 1, ...$ which means that (x_0, y_0) must lie at the centroid of the cell. For simple geometric shapes such as parallelograms, the centroid in fact does coincide with the center; for general shapes they differ. The error that results when it is assumed the two coincide is of higher order in most cases.

VISCOUS TERMS EVALUATION

We now focus on various ways in which the viscous terms of the Navier-Stokes equations may be calculated. Second order derivatives are involved and usually require two separate steps. A first step computes stresses and the second evaluates the viscous terms. Normally viscous terms would require a larger computational molecule than inviscid terms; however most schemes do avoid that large stencil.

The primary considerations of such schemes with regard to the discretization of viscous terms are:

- A small computational molecule and compactness
- accuracy
- cost of computations
- numerical dissipation
- conservation
- free-stream preservation

No one method copes well with all of the above somewhat contradictory considerations.

We now consider various approaches for evaluating viscous terms. Consider the grid shown in Fig. 2 with state variables stored at the cell centers.

In order to evaluate a typical viscous term (e.g.u_{xx}) at a cell center P, an integration is performed over the cell:

$$(u_{xx})_P = \frac{1}{S}\int\int u_{xx}dS = \frac{1}{S}\oint_{ABCD} u_x dy =$$

$$\frac{1}{S}\{(u_x)_{CB}(y_C - y_B) + (u_x)_{DC}(y_D - y_C) + (u_x)_{AD}(y_A - y_D) + (u_x)_{BA}(y_B - y_A)\} \quad (6)$$

where CB, DC, AD, and BA are the points in the middle of the faces of the primary cell $ABCD$ of Fig. 2. The unknown first order derivatives in Eq.(6) then need be evaluated. One approach then performs a second integration over the secondary volume $A'B'C'D'$ [11,13].

$$(u_x)_{CB} = \frac{1}{S'}\int\int_{S'} u_x dS = \frac{1}{S'}\oint_{A'B'C'D'} u dy =$$

$$\frac{1}{S'}\{u_B(y_{B'} - y_{A'}) + u_E(y_{C'} - y_{B'}) + u_C(y_{D'} - y_{C'}) + u_P(y_{A'} - y_{D'})\} \quad (7)$$

where

$$u_C = \frac{1}{4}(u_P + u_E + u_{NE} + u_N),\ u_B = \frac{1}{4}(u_S + u_{SE} + u_E + u_P).$$

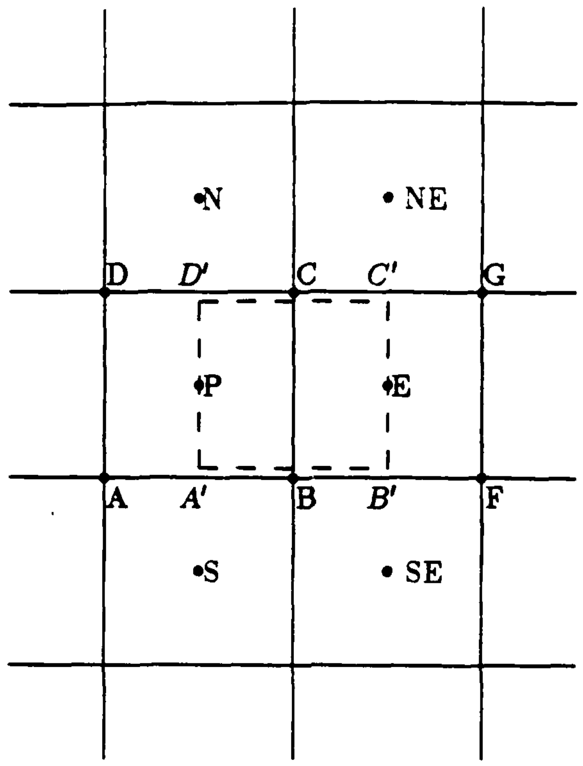


Figure 2: Viscous terms discretization

and S, S' are the areas of the primary and secondary cells respectively. Similar averages are introduced for each of the necessary quantities at the various points. The computational molecule involves 9 points and is also the same molecule that is employed for the discretization of inviscid terms. If the solution is uniform, then $u_C = u_B = u_P = u_E$ and Eq.(7) yields $(u_x)_{CB} = 0$, and consequently $(u_{xx})_P = 0$; which means that the discretization accepts uniform flow as a solution (free-stream preserving).

We now examine whether or not an odd-even mode is accepted by the above discretization. For example, suppose a solution at the grid points N, P, E, NE of a cartesian grid has the form of a sawtooth mode. Then $u_C = u_B = 0$ and the line integration around the secondary cell $A'B'C'D'$ yields $(u_x)_{CB} = 0$. Consequently, $(u_{xx})_P$ vanishes, implying that such a solution is accepted by the scheme.

Another method makes use of averaging in order to obtain the stresses at the cell-faces: $(u_x)_{CB} = \frac{1}{2}\{(u_x)_P + (u_x)_E\}$ (Fig. 2), and then the cells $ABCD$ and $BFGC$ can be employed to calculate $(u_x)_P$ and $(u_x)_E$. This however, results in a large stencil. Numerous other variations can be used, differing mainly in the way the averaging is performed, in order to obtain the required information at various points, and in the way the line integrations are carried out (e.g. midpoint rule, trapezoidal, etc). The use of the small area $ABCD$ around point P (Fig. 2) reduces the computational molecule to only 9 points. The above discretization is also conservative; that is the viscous fluxes cancel one another at the faces of the primary volumes. For example, the flux $F_{CB} = (u_x)_{CB}(y_C - y_B)$ of the volume centered at P, is canceled identically by the flux $F_{BC} = (u_x)_{CB}(y_B - y_C)$ of the volume centered at E.

Another class of methods are those which switch to finite differences when evaluating stresses, instead of using integrations over the secondary cells [2]. For example, the finite-difference value of $(u_x)_{CB}$ on a cartesian mesh, is

$$\frac{u_E - u_P}{x_E - x_P}.$$

Generally for a non-cartesian mesh, the finite-difference expressions for first order derivatives of a quantity U are given by the relations (Fig. 3):

$$\frac{\partial U}{\partial x} = \frac{\Delta_m U.\Delta_l y - \Delta_l U.\Delta_m y}{\Delta_m x.\Delta_l y - \Delta_l x.\Delta_m y}$$

$$\frac{\partial U}{\partial y} = \frac{\Delta_l U.\Delta_m x - \Delta_m U.\Delta_l x}{\Delta_m x.\Delta_l y - \Delta_l x.\Delta_m y}$$

where

$$\Delta_m U = U_E - U_W,\ \Delta_l U = U_N - U_S,$$

etc. The approach remains conservative since the second integration over the primary cells, that is carried out to obtain second order derivatives, is conservative.

So far, we have considered variables to be stored at the centers of the cells. Still another class of methods stores the variables at cell vertices [4,9]. Again primary and secondary control areas are employed in order to compute the viscous terms. One such method will be described in some detail in a later section.

Viscous terms at boundaries require information from outside of the domain in order to be evaluated (pseudo cells) because the boundaries interrupt the primary and secondary cell arrangement. In most cases, however, viscous terms are not required at boundaries since boundary conditions are employed there.

SPATIAL ACCURACY

An evaluation of viscous terms faces more severe accuracy problems than do inviscid terms which consist of lower order derivatives. The higher order derivatives

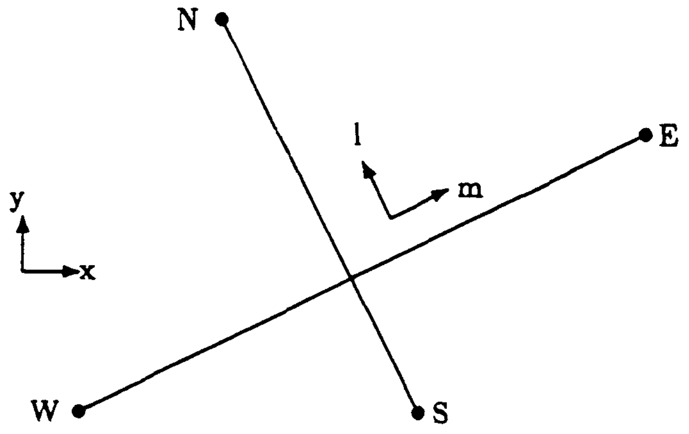


Figure 3: Coordinate definition for non-cartesian mesh

require a broader stencil and the accuracy is more sensitive to grid quality (stretching, skewness, etc). The spatial accuracy of viscous terms discretization therefore, merits special attention.

Two main operations were employed in the previous section when evaluating viscous terms: 1) line integration (with a trapezoidal or midpoint rule), and 2) averaging to obtain values of variables and other geometric quantities at desired locations. Formally, both operations are second order accurate. However, the order depends upon both the grid and the solution. We proceed now to examine various sources of error which can reduce the accuracy of methods.

Integrals Evaluation

Consider the one-dimensional case which evaluates a quantity u at point A ($\overline{u_A}$)using the values of quantities at neighbouring points A^+, A^- (Fig. 4).

$$\overline{u_A} = \frac{1}{\Delta S}\int_{A^-}^{A^+} udS = \frac{1}{\Delta S}\frac{1}{2}(u_{A^-} + u_{A^+})\Delta S \qquad (8)$$

Taylor expanding u_{A^-}, u_{A^+} and substituting in Eq. (8) we obtain:

$$\overline{u_A} = u_A + \frac{1}{2}(\Delta S^+ - \Delta S^-)\frac{\partial u}{\partial s} + \frac{1}{4}((\Delta S^+)^2 + (\Delta S^-)^2)\frac{\partial^2 u}{\partial s^2} + \ldots \Rightarrow$$

$$\overline{u_A} = u_A + \frac{1}{2}(\Delta S^+ - \Delta S^-)\frac{\partial u}{\partial s} + \ldots \qquad (9)$$

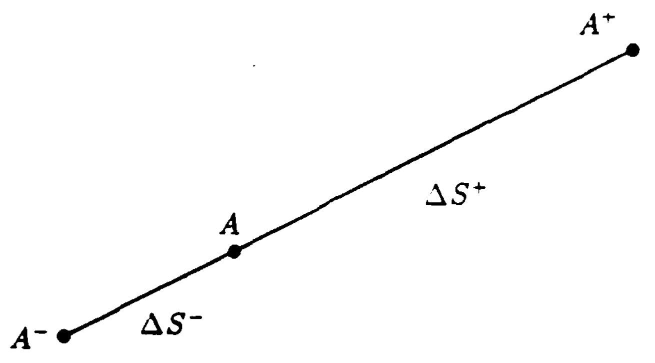


Figure 4: Nonuniform mesh definition

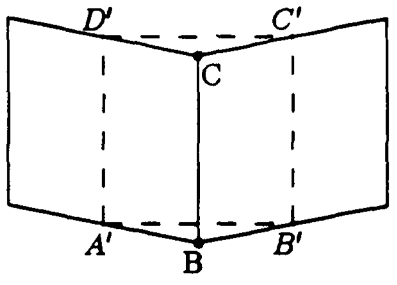


Figure 5: Sheared mesh

The derivative terms on the right hand side represent the error that is made when approximating $\overline{u_A}$. Without stretching ($\Delta S^+ = \Delta S^-$), an evaluation of $\overline{u_A}$ is second order accurate, but reduces to first order when stretching is present. If α is the stretching ratio ($\alpha = \frac{\Delta S^+}{\Delta S^-}$) of a mesh that is stretched exponentially, for the case of a stress (first order derivative), we obtain:

$$\overline{(\frac{\partial u}{\partial s})_A} = (\frac{\partial u}{\partial s})_A + \frac{1}{2}\Delta S^-(\alpha - 1)(\frac{\partial^2 u}{\partial s^2})_A + \dots \qquad (10)$$

If the solution is linear ($u \sim s$), then $(\frac{\partial^2 u}{\partial s^2})_A = 0$ and despite the stretching the evaluation remains second order.

Accuracy is also reduced by a sheared grid (Fig. 5). For a grid with kinks at points C and B, the line integration around the secondary cell $A'B'C'D'$ assumes that straight lines connect the vertices (i.e. point C lies on a line passing through points D' and C').

A more accurate approach would make use of

$$\frac{1}{2}(u_C + u_{C'})\Delta S_{CC'} + \frac{1}{2}(u_{D'} + u_C)\Delta S_{D'C}$$

for the line integration. However, if no stretching is present ($\Delta S_{CC'} = \Delta S_{D'C} = \Delta S/2$), we have instead:

$$\frac{1}{2}(u_{C'} + u_{D'})\frac{\Delta S}{2} + u_C\frac{\Delta S}{2}.$$

Since

$$u_{C'} = \frac{1}{2}(u_{NE} + u_E),\ u_{D'} = \frac{1}{2}(u_N + u_P)$$

there follows

$$\frac{1}{4}(u_P + u_E + u_{NE} + u_N)\frac{\Delta S}{2} + u_C\frac{\Delta S}{2} = u_C\Delta S,$$

as was used here. The way in which the averaging was defined to obtain the quantities at points C,D', and C' eliminates the error for this case.

The line integration around $A'B'C'D'$ is carried out when evaluating $(u_x)_{CB}$ at the middle of face CB. Recall however, that Green's theorem gives the u_x value at the centroid of $A'B'C'D'$, which does not coincide necessarily with the middle of face CB, but introduces an error of higher order than that induced by stretching. Finally, skewness of the cell does not affect the accuracy of a line integration.

It is apparent that errors due to stretching are the most serious; however several approaches can be adopted to compensate. Usually this involves weighting factors based on the grid stretching [14]. Criticism for such approaches includes the cost of extra computations, the treatment at boundaries (either flow or zonal), extra storage for weighting factors, and the use of information from outside of each cell, which is undesirable when unstructured grids are present.

Averaging

Averaging is used frequently during the numerical processing of viscous terms. Simple algebraic averaging is formally of second order and degrades to first order when stretching is present. The algebraic averaging when obtaining u_P is second order only if

$$\vec{r}_P = \frac{1}{4}(\vec{r}_1 + \vec{r}_2 + \vec{r}_3 + \vec{r}_4)$$

where $\vec{r}_i$ are the position vectors of the corner points of a cell, and $\vec{r}_P$ is the corresponding position vector for an interior point P. Area weighted averages and interpolation formulas which use splines can be employed. Problems associated with such approaches are similar to those mentioned above.

As was mentioned, other than grid quality, the factor which most influences accuracy is the form of the solution itself. The more non-linear the solution, the larger the errors. For a typical boundary layer profile, the outer edge region is sufficiently non-linear to cause accuracy problems, in comparison to the inner portion of the profile which is relatively linear.

Another factor which introduces errors is artificial viscosity. The order of the usual second and fourth order smoothing operators on smooth grids reduces for non-uniform grids as we will see in a later section. Lastly, all of the above errors scale with the mesh size. Both the integration and averaging errors are more significant in the regions where the grid is coarse.

Limitations on higher order numerical modeling

To date, there have been very few finite volume schemes for Navier-Stokes equations (especially for compressible flows) that correspond to higher than second order of accuracy. There are a few possible reasons for that. Higher order schemes involve more computations which may outweigh the advantage of using coarser grids. A larger computational molecule usually is required and this poses problems in the boundary treatment. Higher order accuracy often leads to tighter stability restrictions. Lastly, extra storage then may be needed, and care must be taken to account for higher than second order errors. These can still be of lower order than the method, and have been ignored in the present schemes (e.g. the difference between the centroid and center of a cell).

VISCOUS GRID

The issue of spatial accuracy due to viscous term discretization has been shown to depend substantially upon the grid quality. On the other hand, the accuracy of inviscid term evaluations is generally much more insensitive to the grid properties (stretching, shearing). The second most important factor determining accuracy is the solution itself. The larger its derivatives, the bigger is the error. Gradients in viscous regions are generally much larger than those in inviscid regions (excluding typical discontinuities). It should be apparent that a distinction must be made between viscous and inviscid grids. We now focus on the specific grid requirements in viscous regions.

Number of nodes across the shear layer

Usually, a shear layer requires $O(10)$ to $O(10^2)$ points, for example, a number like 15 is common practice. If the Reynolds number (Re) is small, the layer is relatively large, and the spacing between grid points is large (coarse grid). A larger

Re, implies finer viscous grids. Many points are wasted in a gradual distribution from viscous to outer inviscid regions, given the stretching restriction. The 15 points in the case of a laminar layer are placed in a region with thickness $\delta \sim Re^{\frac{1}{2}}$, while in a turbulent layer $\delta \sim Re^{\frac{4}{5}}$. Moreover, about two points are needed in the viscous sublayer of a turbulent shear layer, which implies that the spacing at the wall Δy_{min} should be such that

$$\Delta y^{+}_{min} \equiv \frac{u^{+} \Delta y_{min}}{\nu} \sim 2 \ , u^{+} \equiv \sqrt{\frac{\tau}{\rho}}.$$

Approximating the wall stress τ with $\frac{\Delta u}{\Delta y_{min}} = \frac{u}{\Delta y_{min}}$, and nondimensionalizing Δy_{min} with a reference length L, results in

$$\Delta y_{min} \sim \frac{1}{Re}.$$

<u>Stretching</u>

Equation (10) shows the lowest order error term to be $\frac{1}{2}\Delta S(\alpha - 1)\frac{\partial^2 u}{\partial s^2}$, which should be much smaller than the first term $\frac{\partial u}{\partial s}$. In the linear region of a typical shear layer profile $\frac{\partial^2 u}{\partial s^2}$ is relatively small and α can be much bigger than the stretching that is allowed in the edge region, where $\frac{\partial u}{\partial s}$ is larger. Hopefully the above term is at least an order of magnitude smaller than the first term u:

$$\frac{1}{2}\Delta S(\alpha - 1)\frac{\partial^2 u}{\partial s^2} < \frac{1}{10}(\frac{\partial u}{\partial s}).$$

Linearizing the derivative and assuming $\Delta u \sim u$, implies that:

$$\alpha < 1.2$$

This suggests that the maximum allowable stretching $(\alpha - 1)$ is 20%. This has been empirically considered to be an upper limit for grid stretching, although in many cases, this value is exceeded.

<u>Directionality</u>

Generally, in a shear layer there are large differences in magnitude between gradients in streamwise and normal directions. I.e, $\frac{\partial u}{\partial x} \ll \frac{\partial u}{\partial y}$, or $\frac{\Delta u}{\Delta x} \ll \frac{\Delta u}{\Delta y}$ and this implies $\Delta y \ll \Delta x$. Such directionality in the solution 'imposes' a similar constraint on the grid. Viscous cells usually have large aspect ratios which can be of the order of 100 in many cases. The directionality requirement is met much more easily by quadrilateral rather than triangular meshes. This is a dominant reason for the absence of triangles in viscous regions computations.

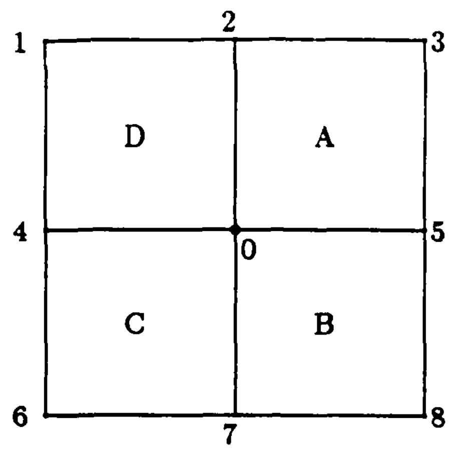


Figure 6: Node and cell designations for smoothing operators

Adaptive grids

The resolution and stretching requirements impose serious problems on viscous grid generation. The allowed stretching together with the smallest spacing at the wall, result in excess resolution in the inviscid region. Also, the location and thickness of viscous regions are not known a priori. This results in considerable empiricism in grid construction and its later modification upon examining the solution. Adaptive locally embedded viscous grids provide local refinement using an adaptive algorithm which senses the viscous regions during the course of the computation [4,5,3]. A detailed examination of such methods is given in the chapter on adaptation methods for viscous flows.

ARTIFICIAL DISSIPATION

Smoothing that is accomplished by explicitly adding dissipation is employed by the vast majority of existing schemes, especially those concerned with compressible flows. There are two main types of such artificial smoothing: one is used to capture shocks, the second is designed to damp spurious oscillations throughout the field and to suppress odd-even decoupling of the solution. These second and fourth order smoothing operators are examined in detail in order to address the relevant issues of viscous region contamination by artificial smoothing and the degradation of order with grid stretching.

Second order (shock) smoothing

Second order smoothing provides damping necessary to smear a shock, which ideally has zero thickness, in such a way that oscillations are avoided. Since it is required only in shock regions, a switch is employed to turn it off elsewhere.

In two dimensions, the damping term may have the form:

$$\left|\frac{\partial P}{\partial x}\right| \cdot \frac{\partial u}{\partial x} + \left|\frac{\partial P}{\partial y}\right| \cdot \frac{\partial u}{\partial y} \tag{11}$$

Consider the specific discretization for a cell-vertex scheme at node 0 (Fig. 6).

The node receives contributions from each of the four surrounding cells; That from cell A is:

$$\begin{aligned} S^{(2)}_{0A} = & \left|\frac{(P_3+P_5)-(P_2+P_0)}{P_3+P_5+P_2+P_0}\right| .\{(u_3+u_5) \quad - \quad (u_2+u_0)\} + \\ & \left|\frac{(P_2+P_3)-(P_0+P_5)}{P_2+P_3+P_0+P_5}\right| .\{(u_2+u_3) \quad - \quad (u_0+u_5)\} \end{aligned} \tag{12}$$

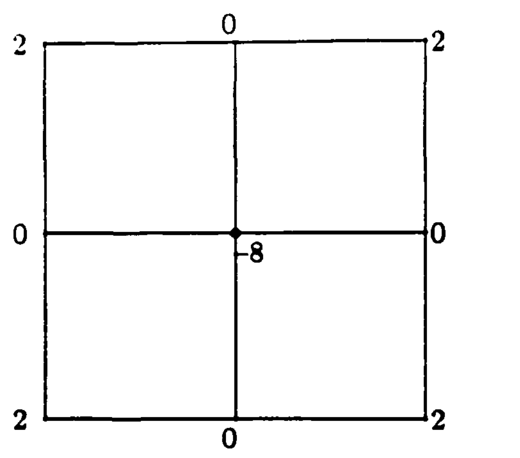
Similarly, from cell D it is:

$$\begin{aligned} S^{(2)}_{0D} = & \left|\frac{(P_1+P_4)-(P_2+P_0)}{P_1+P_4+P_2+P_0}\right| .\{(u_1+u_4) \quad - \quad (u_2+u_0)\} + \\ & \left|\frac{(P_1+P_2)-(P_4+P_0)}{P_1+P_2+P_4+P_0}\right| .\{(u_1+u_2) \quad - \quad (u_4+u_0)\} \end{aligned} \tag{13}$$

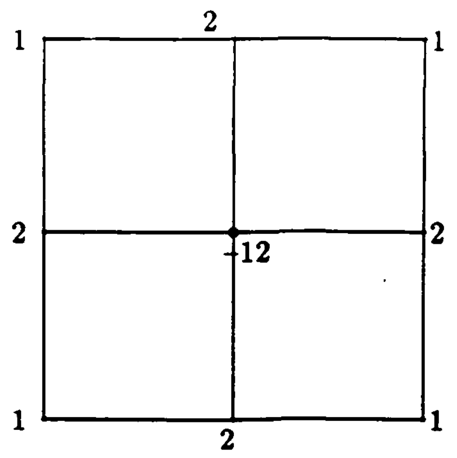
Similar expressions furnish contributions from cells C and B. Pressure differences in the switch are normalized by the sum of the pressures at the four corners of each cell. The sum of the smoothing distributions to the four nodes of each cell is zero, which implies that the above operator is conservative. For the special case of $\frac{\partial P}{\partial x} = \frac{\partial P}{\partial y}$, the computational molecule of the smoothing operator is the one shown in Fig. 7 (a).

A question does arise with respect to the appropriate order of the pressure switch. The purpose of the switch is to act on only large gradients (shocks) and not on those which are small. The larger the stencil of the switch, the more likely it is to 'pick' relatively mild gradients which do not actually represent shocks. Moreover, a higher order switch is computationally expensive. A proof follows which uses Burgers equation $uu_x = \nu u_{xx}$ applied to a shock region of thickness δ. From this relation, it follows that $\frac{u^2}{\delta} \sim \frac{\nu u}{\delta^2} \Rightarrow \nu \sim u.\delta$, which implies that the bigger the smoothing coefficient, the more smeared the shock region will be. Continuing from the previous relation, $\nu \sim \Delta u.\Delta x$ but $\Delta u \sim \Delta P$, thus

$$\nu \sim \Delta P.\Delta x$$



(a) Second order smoothing operator



(b) Laplacian operator

Figure 7: Smoothing stencils

which implies that the pressure switch should be a *first* difference in pressure.

Fourth order smoothing

The fourth order smoothing that is used away from shocks to suppress odd-even modes and damp spurious oscillations is turned off near shocks because it is destabilizing. The operator is formed in two steps. The second order difference operator is formed in the first step (Fig. 6):

$$\begin{aligned} D^2_{0A} &= u_0 + u_2 + u_3 + u_5 - 4u_0 \\ D^2_{0B} &= u_7 + u_0 + u_5 + u_8 - 4u_0 \\ D^2_{0C} &= u_6 + u_4 + u_0 + u_7 - 4u_0 \\ D^2_{0D} &= u_4 + u_1 + u_2 + u_0 - 4u_0 \end{aligned} \tag{14}$$

Summing up contributions from the four cells surrounding node 0 results in a Laplace stencil (Fig. 7 (b)).

The second step duplicates the first, replacing state variables by second order differences from the first step.

$$\begin{aligned} -D^4_{0A} &= D^2_0 + D^2_2 + D^2_3 + D^2_5 - 4D^2_0 \\ -D^4_{0B} &= D^2_7 + D^2_0 + D^2_5 + D^2_8 - 4D^2_0 \\ -D^4_{0C} &= D^2_6 + D^2_4 + D^2_0 + D^2_7 - 4D^2_0 \\ -D^4_{0D} &= D^2_4 + D^2_1 + D^2_2 + D^2_0 - 4D^2_0 \end{aligned} \tag{15}$$

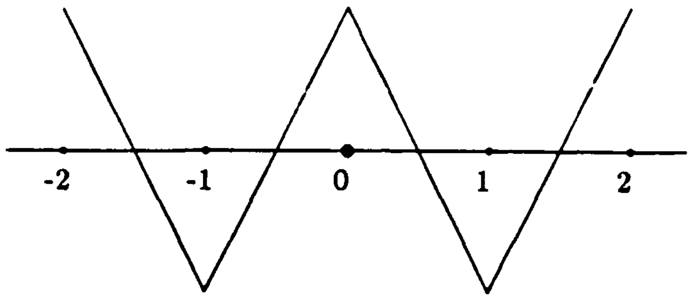


Figure 8: Sawtooth mode

It is illustrative to study how the smoothing operator acts to remove an odd-even mode for the one-dimensional case shown in Fig. 8. The operator in one-dimensional form is:

$$-D_0^4 = u_{-2} - 4u_{-1} + 6u_0 - 4u_1 + u_2, \tag{16}$$

where $-2, -1, 0, 1, 2$ are neighbouring points (Fig. 8). Substitution of sawtooth mode values in Eq. (16) shows that D_0^4 furnishes such a contribution to node 0 as to reduce the odd-even mode u_0.

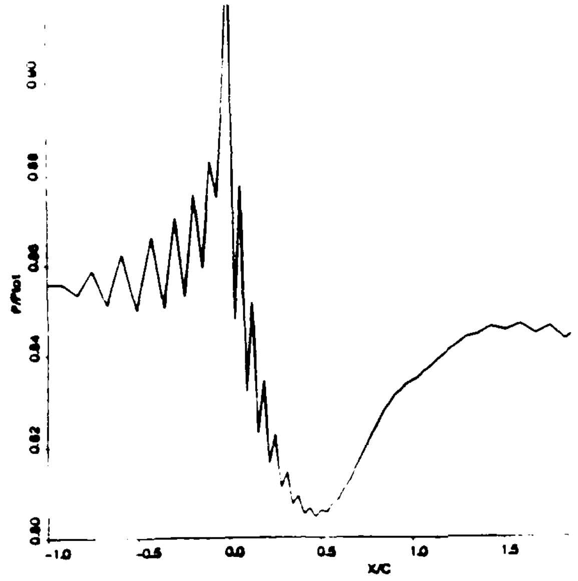
Combined Second and Fourth order smoothing

The combined second and fourth order smoothing operator has the following form:

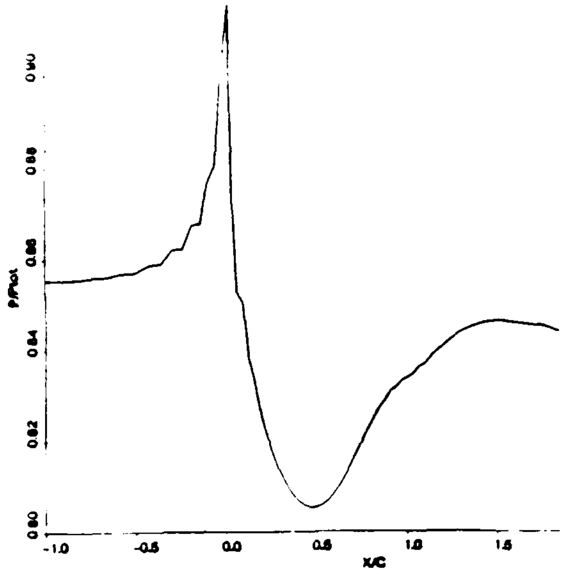
$$\delta u_i = \frac{1}{4}\sigma_2 S_i^2 - \frac{1}{4}max(0, \sigma_4 - \sigma_2 \Delta P).D_i^4, \tag{17}$$

where S_i^2, D_i^4 are the second and fourth order operators discussed above, σ_2, σ_4 are corresponding smoothing coefficients, and ΔP is the pressure switch. Near shocks, the term $\sigma_2 \Delta P$ dominates over σ_4 and therefore, $max(0, \sigma_4 - \sigma_2 \Delta P)$ vanishes and switches off the fourth order smoothing.

Numerical experiments have been carried out with a node-based Navier-Stokes scheme (described in the next section), to determine optimum values for the smoothing coefficients. The example considered flow at Mach number 0.5 and 1.4 in a channel with a bump and a 65 × 33 grid. Figures 9a,b show C_P distributions for the subsonic case with and without fourth order smoothing only and it is clear that even a very small amount of dissipation suppresses the sawtooth mode. Figure 10 demonstrates contamination of the boundary layer by smoothing via wall skin friction coefficient distributions for values of the smoothing coefficient equal to 0.004



(a) without smoothing



(b) with smoothing $\sigma_4 = 0.0005$

Figure 9: Effect of fourth order smoothing on odd-even decoupling

and zero. Similarly, a supersonic case was used to study the second order smoothing coefficient. Figures 11a,b show C_P wall distributions with fourth order smoothing coefficient $\sigma_4 = 0.0005$ and shock smoothing coefficients σ_2 of 0.40 and 0.05. Clearly the value of 0.40 smears the shock excessively, while the 0.05 value is too small and pre-shock oscillations appear. A reasonable choice of smoothing coefficients is: $\sigma_2 = 0.20$ and $\sigma_4 = 0.0004$.

Boundary layer contamination

Smoothing is required mainly in the inviscid regions; very little is needed in viscous regions. The basic requirement is suppression of odd-even decoupling. The presence of physical viscosity does not suppress such decoupling since it is allowed by the viscous terms discretization, as was shown in a previous section. Nevertheless, the presence of smoothing in viscous regions may seriously deteriorate accuracy. The way in which smoothing terms affect the viscous layer solution as well as the resolution requirements to avoid the error, are of interest.

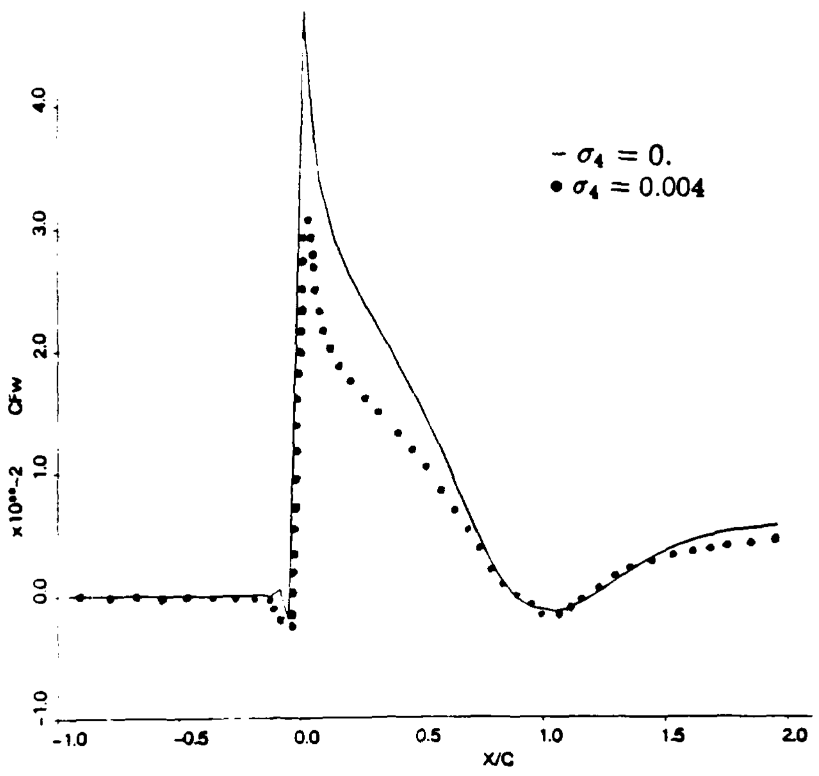
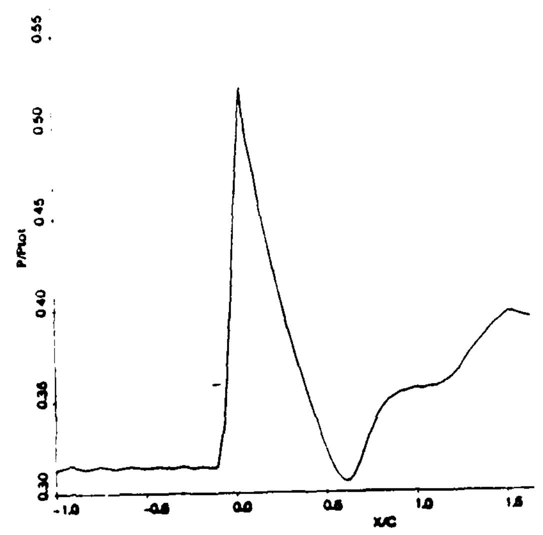
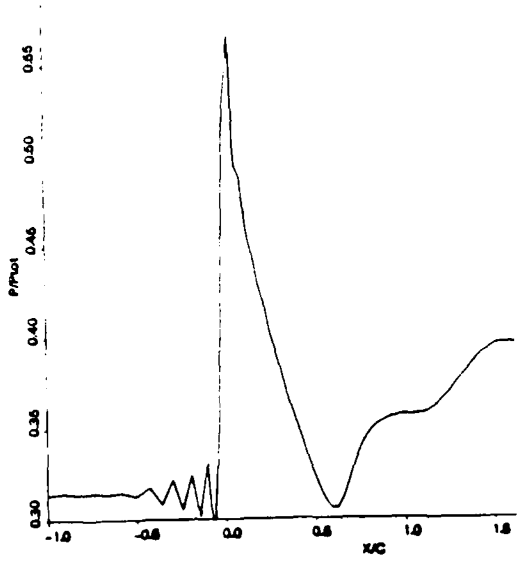


Figure 10: Contamination of boundary layer solution by smoothing



(a) **smoothing coeff.** $\sigma_2 = 0.40$



(b) **smoothing coeff.** $\sigma_2 = 0.05$

Figure 11: Shock capturing with second order smoothing

The second and fourth order derivatives usually take large values within the viscous region and therefore smoothing contaminates the viscous layer more than other regions. For the model diffusion equation $u_t = \nu u_{yy}$, we have:

$$u_t = \nu u_{yy} + \frac{|u| + c}{CFL}\sigma_2 \delta P(\Delta y) u_{yy} + \frac{|u| + c}{CFL}\nu_4 (\Delta y)^3 u_{yyyy}, \tag{18}$$

where the last two terms represent first and third order errors introduced by the two smoothing operators. A CFL stability condition $(\frac{(|u|+c)\Delta t}{\Delta x} = CFL)$ eliminates the time-step Δt from the above expression. To ensure that artificial viscosity is much less than the physical viscosity: $\frac{|u|+c}{CFL}\sigma_2 \delta P \Delta y \ll \nu$, which yields

$$\sigma_2 \ll \frac{CFL}{Re.\delta P.\Delta y}. \tag{19}$$

in which $Re \equiv \frac{(|u|+c)L}{\nu}$. If ΔP is negligible across the shear layer, then relatively large values of σ_2 can be used. For example, for the case $CFL = 1, Re = 10^6, \delta P = 0.1, \Delta y = 10^{-4}$, it is obtained that $\sigma_2 \ll 0.1$.

For a fixed value of the smoothing coefficient σ_2, the resolution of the shear layer should be such that: $O(\frac{\nu u_{yy}}{\frac{|u|+c}{CFL}\sigma_2 \delta P \Delta y u_{yy}}) >> 1 \Rightarrow$

$$N \equiv \frac{\delta}{\Delta y} >> \frac{Re_\delta \sigma_2 \delta P}{CFL}, \tag{20}$$

N being the number of grid points across the shear layer and $Re_\delta = \frac{(|u|+c)\delta}{\nu}$. For the case of $CFL = 1, Re_\delta = 10^3, \delta P = 0.1, \sigma_2 = 0.1$, the number of points within the layer should be $N >> 10$ so that the real viscous terms dominates. Re_δ is based on $|u| + c$, and therefore in nearly incompressible cases it can take very large values, which implies that considerable resolution is needed.

For a fixed value of the smoothing coefficient ν_4, the resolution of the shear layer should be such that: $O(\frac{\nu u_{yy}}{\frac{|u|+c}{CFL}\nu_4 (\Delta y)^3 u_{yyyy}}) >> 1 \Rightarrow$

$$N >> (\frac{Re_\delta \nu_4}{CFL})^{\frac{1}{3}}. \tag{21}$$

Comparison with relation (20), shows that this resolution requirement is much less severe, as a result of the higher order of this operator. For example, if $CFL = 1, Re_\delta = 10^3, \nu_4 = 0.001$, the number of points within the layer $N >> 1$.

Several suggestions have been made in order to avoid a deterioration of accuracy due to smoothing. One gradually reduces smoothing on approaching the wall by reducing the values of the smoothing coefficients [2]. However, in many cases the most serious contamination of the boundary layer does not occur at the wall but in regions with larger u_{yy}, u_{yyyy} gradients. Another method employs directional smoothing by applying the operator to the streamwise direction only. This requires a quadrilateral mesh, or triangular with a quadrilateral structure (e.g. triangles which are formed by dividing quadrilaterals along a diagonal) [16]. A third approach applies resolution in the viscous regions by means of local embedding (cell division), which results in a reduction of cell size and therefore a reduction in the magnitude of the smoothing error. Adaptive algorithms place finer grids in those regions with higher gradients where smoothing is larger.

Grid stretching increases smoothing error

The above smoothing operators introduce first and third order errors only for uniformly spaced grids. The actual order can be demonstrated by considering a one-dimensional stretched grid. A second order smoothing operator without the pressure switch has the form: $S_0^2 = u_1 - 2u_0 + u_{-1}$.

A Taylor series expantion about point 0, leads to: $S_0^2 = (h^+ - h^-)u_x + \frac{1}{2}\{(h^+)^2 + (h^-)^2\}u_{xx} + \ldots$. Assumming exponential stretching, and with $\alpha \equiv \frac{h^+}{h^-}$, the operator becomes:

$$S_0^2 = h^-(\alpha - 1)u_x + \frac{1}{2}(h^-)^2(\alpha^2 + 1)u_{xx} + \ldots \qquad (22)$$

The first order term $h^-(\alpha - 1)u_x$ appears, which increases the smoothing error and makes the operator *dispersive* rather than *dissipative*. Similarly, the fourth order difference operator $D_0^4 = u_{-2} - 4u_{-1} + 6u_0 - 4u_1 + u_2$ has the form for an exponentially stretched mesh:

$$D_0^4 = (\alpha - 2)(\alpha - 1)h^- u_x + 2\alpha(1 + \frac{\alpha}{4})(\alpha^2 + 1)(h^-)^2 u_{xx} + \ldots \qquad (23)$$

Again, the first order term $(\alpha - 2)(\alpha - 1)h^- u_x$ increases the error and makes smoothing *dispersive*. It is to be noted that exponentially stretched meshes are widely employed for viscous computations. The dissipative terms in Eqns. (22),(23) are positive for any α which precludes the possibility of having negative damping for some α.

AN EXPLICIT NODE-BASED FINITE-VOLUME SCHEME

A conservative finite-volume scheme developed for the Navier-Stokes equations in [4] will be described. It is an explicit, time-marching scheme with the state-

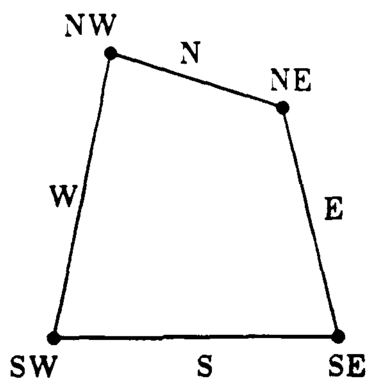


Figure 12: Cell node and face designations

variables stored at grid nodes. All necessary operations are completed within the cell, which makes the scheme suitable for unstructured (locally embedded) meshes. The discretization of the inviscid terms will be presented, then the treatment of the viscous terms will be described in detail, and finally the code is examined in terms of computer requirements and example cases.

Inviscid terms

A one-step Lax-Wendroff-type integration scheme [10] was employed for treatment of the convective terms of the Navier-Stokes system. The corresponding integral relation over a cell is $\frac{\partial}{\partial t}\int\int_{cell\,area} U dx dy + \oint_{cell\,faces}(F dy - G dx) = 0$. The first term, representing the change in time of U over the cell area S, is discretized as $\frac{\partial U_C}{\partial t}.S$. The second term, representing the convective fluxes across the cell-faces, is computed via the trapezoidal integration rule. Then (Fig. 12)

$$\begin{aligned}\frac{\partial U_C}{\partial t}.S &+ \frac{F_{SW}+F_{SE}}{2}(y_{SW}-y_{SE}) - \frac{G_{SW}+G_{SE}}{2}(x_{SW}-x_{SE}) \\ &+ \frac{F_{NW}+F_{SW}}{2}(y_{NW}-y_{SW}) - \frac{G_{NW}+G_{SW}}{2}(x_{NW}-x_{SW}) \\ &+ \frac{F_{NE}+F_{NW}}{2}(y_{NE}-y_{NW}) - \frac{G_{NE}+G_{NW}}{2}(x_{NE}-x_{NW}) \\ &+ \frac{F_{SE}+F_{NE}}{2}(y_{SE}-y_{NE}) - \frac{G_{SE}+G_{NE}}{2}(x_{SE}-x_{NE}) \qquad (24)\end{aligned}$$

The above change in time over the cell ($\frac{\partial U_C}{\partial t}.S$) must be distributed to its four nodes according to:

$$
\begin{aligned}
(\delta U)_{SW} &= \frac{1}{4}\{\Delta U_C - \Delta f_C - \Delta g_C\} \\
(\delta U)_{NW} &= \frac{1}{4}\{\Delta U_C - \Delta f_C + \Delta g_C\} \\
(\delta U)_{NE} &= \frac{1}{4}\{\Delta U_C + \Delta f_C + \Delta g_C\} \\
(\delta U)_{SE} &= \frac{1}{4}\{\Delta U_C + \Delta f_C - \Delta g_C\}
\end{aligned}
\tag{25}
$$

where

$$
\begin{aligned}
\Delta f_C &= \frac{\Delta t}{\Delta s}(\Delta F_C \Delta y^l - \Delta G_C \Delta x^l) \\
\Delta g_C &= \frac{\Delta t}{\Delta s}(\Delta G_C \Delta x^m - \Delta F_C \Delta y^m)
\end{aligned}
$$

$$\Delta F = (\frac{\partial F}{\partial U})\Delta U, \ \Delta G = (\frac{\partial G}{\partial U})\Delta U$$

and

$$
\begin{aligned}
\Delta x^l &= 0.5(x_{NW} + x_{NE} - x_{SW} - x_{SE}) \\
\Delta y^l &= 0.5(y_{NW} + y_{NE} - y_{SW} - y_{SE}) \\
\Delta x^m &= 0.5(x_{NE} + x_{SE} - x_{NW} - x_{SW}) \\
\Delta y^m &= 0.5(y_{NE} + y_{SE} - y_{NW} - y_{SW}).
\end{aligned}
$$

Here ΔU_C is the sum of the flux terms in Eq.(24).

Viscous terms

The viscous part of the integral Navier-Stokes equations is

$$\int\int \frac{\partial U}{\partial t} dxdy = \oint (Rdy - Sdx) \tag{26}$$

and is discretized by using two different cells. One, the *secondary* cell, is used to evaluate stresses (first order derivatives), and another, the *primary* cell, is employed to calculate the viscous terms (second order derivatives).

As was mentioned, a main concern when discretizing viscous terms is to keep the stencil small. Figure 13 shows the *primary* cell $ABCD$ that is used to compute viscous terms at node O.

The unsteady term $\int\int \frac{\partial U}{\partial t} dxdy$ is discretized as $\frac{\partial}{\partial t}\int\int U dxdy = \frac{\partial}{\partial t} U_O.S = \frac{\Delta U}{\Delta t}.S$, where S is the area of the *primary* cell $ABCD$. Therefore, the entire integral equation takes the discrete form over $ABCD$:

$$
\begin{aligned}
\Delta U = \frac{\Delta t}{S}\oint_{ABCD}(Rdy - Sdx) \\
= \frac{\Delta t}{S}.\{ \; & + \; R_{BA}.\Delta y_{BA} - S_{BA}.\Delta x_{BA} \\
& + \; R_{CB}.\Delta y_{CB} - S_{CB}.\Delta x_{CB} \\
& + \; R_{DC}.\Delta y_{DC} - S_{DC}.\Delta x_{DC} \\
& + \; R_{AD}.\Delta y_{AD} - S_{AD}.\Delta x_{AD}\}
\end{aligned} \qquad (27)
$$

The terms $R_{BA}, S_{BA}, R_{CB}, S_{CB}$, *etc* are stress and heat conduction terms that should be evaluated at cell-faces. The *secondary* cell (defined by the points BA, B', C', DC, Fig. 13) is employed for the evaluation of first order derivatives. A typical derivative, $\frac{\partial U}{\partial x}$, is evaluated as follows:

$$
\frac{\partial U}{\partial x} = \frac{1}{S_{CB}}\int\int \frac{\partial U}{\partial x}dS = \frac{1}{S_{CB}}\oint Udy =
$$

$$
\begin{aligned}
= \frac{1}{S_{CB}}.\{U_B(y_{B'} - y_{BA}) \; &+ \; U_E(y_{C'} - y_{B'}) + \\
U_C(y_{DC} - y_{C'}) \; &+ \; U_O(y_{BA} - y_{DC})\}
\end{aligned} \qquad (28)
$$

This viscous terms treatment is second order accurate in space, and first order accurate in time for uniform meshes. Stretching and skewness introduce errors when evaluating line integrals as was discussed. On a cartesian mesh, the above spatial discretization reduces to classical central differencing. The choice of *secondary* cells plays a crucial role in the behaviour of the scheme for odd-even decoupling of the solution. The above choice of *asymmetric* overlapping *secondary* cells has the effect of suppressing the one and two-directional odd-even modes shown in Fig. 14, where $+$ and $-$ indicate deviations from a mean.

In the first case the integration makes a positive contribution to node O, while the second provides a negative contribution. Therefore, in both types of odd-even modes, the scheme tends to suppress. Instead of a staggered *secondary* cell, another possibility would use averaging in order to obtain the face stresses. For the CB node, the averaging $R_{CB} = \frac{1}{2}(R_C + R_B)$ can be used. In this case, the grid cells B, C are used as *secondary* cells in order to evaluate R_B, R_C. However, this method allows the two-directional odd-even mode to exist, since it contributes zero to node O, and thus does not cancel the sawtooth mode.

Equations (27),(28) can be manipulated in such a way that all necessary operations can be performed within each grid cell without using any outside information.

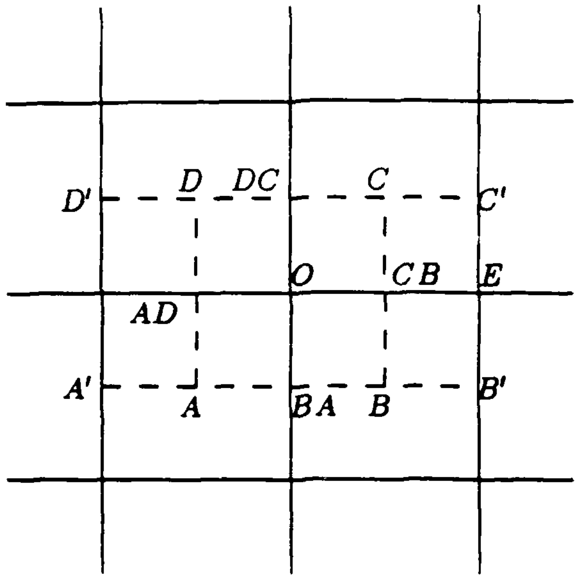


Figure 13: Viscous terms discretization

The terms in (27) are split as follows:

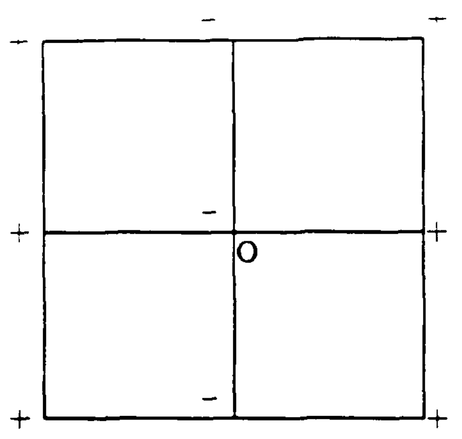
$$\begin{aligned}\Delta U = \frac{\Delta t}{S}.\{ \; & + \; (R_B + R_A).\Delta y_{BA} - (S_B + S_A).\Delta x_{BA} \\ & + \; (R_C + R_B).\Delta y_{CB} - (S_C + S_B).\Delta x_{CB} \\ & + \; (R_D + R_C).\Delta y_{DC} - (S_D + S_C).\Delta x_{DC} \\ & + \; (R_A + R_D).\Delta y_{AD} - (S_A + S_D).\Delta x_{AD} \; \} \qquad (29)\end{aligned}$$

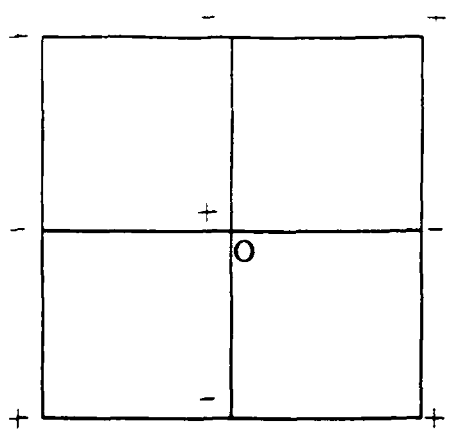
Each of the terms (e.g. R_B, S_B, R_A, S_A,etc) contains information only from cells B, A respectively. It is assumed that the term $\frac{\Delta t}{S}$ is the same for the cells A, B, C, D. Also, $\Delta y_{BA} \equiv y_B - y_A \approx y_{B'} - y_{BA} \approx y_{BA} - y_{A'}$. For a stretched mesh, the above assumptions introduce errors which are of the same order as those introduced by the original line integrations around the *primary* and *secondary* cells. The following example illustrates how the stress and heat conduction terms R, S are split, for the case of the derivative $(u_x)_{CB}$. One obtains:

$(u_x)_{CB} =$

$$\begin{aligned}&\frac{1}{S_{CB}}.\{U_E(y_{C'} - y_{B'}) + U_C(y_{DC} - y_{C'}) + U_O(y_{BA} - y_{DC}) + U_B(y_{B'} - y_{BA})\} \\ &\qquad = \frac{1}{S_C}.\{U_E(y_{C'} - y_E) + U_C(y_{DC} - y_{C'}) + U_O(y_O - y_{DC})\} \; + \\ &\qquad\quad \frac{1}{S_B}.\{U_O(y_{BA} - y_O) + U_B(y_{B'} - y_{BA}) + U_E(y_E - y_{B'})\} \qquad (30)\end{aligned}$$

where $S_{CB} \approx S_C \approx S_B$.





(a) one direction mode (b) two directions mode

Figure 14: Suppression of odd-even modes

Equation (29) gives the contributions to node O by its surrounding cells A, B, C, D. The contributions of each cell to its four corners (Fig. 12) follow from equation (29):

$$\begin{aligned}
(\Delta U)_{SW} &= \frac{\Delta t}{S}.\{(+R_S\Delta y^l - R_W\Delta y^m) - (+S_S\Delta x^l - S_W\Delta x^m)\} \\
(\Delta U)_{NW} &= \frac{\Delta t}{S}.\{(+R_N\Delta y^l + R_W\Delta y^m) - (+S_N\Delta x^l + S_W\Delta x^m)\} \\
(\Delta U)_{NE} &= \frac{\Delta t}{S}.\{(-R_N\Delta y^l + R_E\Delta y^m) - (-S_N\Delta x^l + S_E\Delta x^m)\} \\
(\Delta U)_{SE} &= \frac{\Delta t}{S}.\{(-R_S\Delta y^l - R_E\Delta y^m) - (-S_S\Delta x^l - S_E\Delta x^m)\} \qquad (31)
\end{aligned}$$

where R_S is the part of the split stress which corresponds to the south face of the cell and similarly for the other subscripts N, W, S. The above contributions to the four corners sum up to zero, which implies that the scheme is conservative.

Smoothing is required for the inviscid region, and the operators described in the previous section were applied. Ni's inviscid multigrid operator [10] was also applied to accelerate the wave propagation and convergence in the inviscid region, but it is not really effective in a shear layer. There, the solution advances to steady-state at a slower rate due to small time-steps, as is evident in the later iterations in Fig. 15. Initially, there is an acceleration towards convergence within the inviscid region, followed by the largest errors persisting within the boundary layer region. A typical acceleration factor is approximately five.

The solver requires storage of both state variables and their changes in time (δU) at each node, as well as viscosity and time-step values. The code takes 0.006 seconds per node per iteration in CPU time on a Vax/750 computer and has been

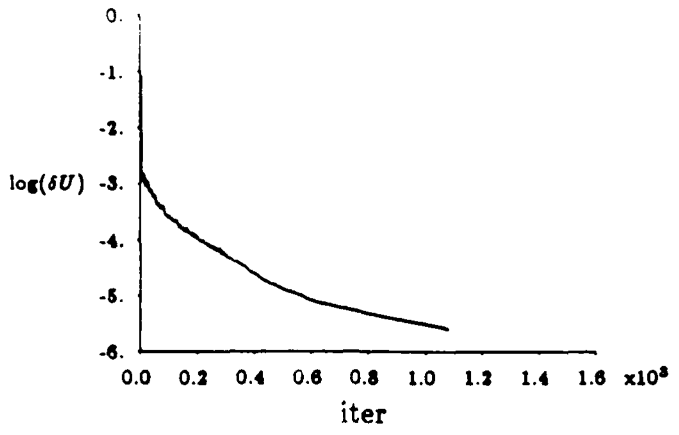


Figure 15: Convergence history with multigrid

vectorized for the Alliant/Fx8 mini-supercomputer to attain a speed-up factor of about 25.

Example cases

Some examples of computations with the previously described scheme follow.

The first case is a 10% circular arc cascade in a subsonic flow [4] of $M_\infty = 0.5$ and $Re = 8 \times 10^3$ (Fig. 16(a)) with a 65×33 grid. The C_f curve is compared with [2,1] in Fig. 16(b). The second case is a 8% circular arc cascade with $M_\infty = 1.4$ and $Re = 23 \times 10^3$ [4]. Figure 17 illustrates the flow field. An oblique shock forms at the leading edge and is reflected at the upper symmetry boundary. The reflected shock then interacts with the boundary layer at the trailing edge region, which separates and reattaches downstream.

Another case is a RAE 2822 airfoil in transonic flow of $M_\infty = 0.75$ and $Re = 6.2 \times 10^6$ and 2.70° angle of attack with a C-grid of 129×49. Fig. 18 shows Mach number contours indicating a normal shock at the suction side which interacts with the boundary layer. The case of a NACA 0012 airfoil in subsonic flow of $M_\infty = 0.5$ with $Re = 2.91 \times 10^6$ and 1.77° angle of attack together with a comparison to experiment, is presented in the chapter on adaptation methods for viscous flows.

CONCLUDING REMARKS

We have considered the finite-volume approach and its use to solve the Navier-Stokes equations with an emphasis on discretization of the viscous terms. The use of secondary cells in order to reduce the stencil and to ensure a compact scheme is the most common approach. The choice of the cell arrangement for the evaluation of derivatives is crucial for the suppression of odd-even modes and, therefore, for the enhancement of convergence and reduction of artificial dissipation. Grid quality challenges the accuracy of most of the current schemes, especially in viscous regions. There, the grid should be constructed carefully with respect to the maximum allowed stretching and resolution. Adaptive grids can offer flexibility in viscous grid generation. The use of artificial dissipation, especially on stretched grids, can deteriorate the accuracy of viscous solutions, if care is not taken in reducing it in shear layer regions. The current Navier-Stokes codes have been successful in predicting quite complicated flows such as those involving shock boundary layer interactions. However, their relative low order of accuracy requires excess resolution and the number of required operations per grid node is relatively large, resulting in quite expensive computations. Adaptive algorithms, which use grids and equations that recognize and follow the flow physics during a computation, seem to be promising for future routine use of current finite-volume Navier-Stokes schemes in engineering applications.

Acknowledgement This work was supported by AFOSR Grant 82-0136 , Drs J.D. Wilson, J. McMichael, and L. Sakell serving as technical monitors.

References

[1] R. V. Chima. Efficient Solution of theEuler and Navier-Stokes Equations with a Vectorized Multiple-Grid Algorithm. *AIAA Journal*, 23:23–32, Jan 1985.

[2] R. L. Davis, Ron-Ho Ni, and J.E. Carter. *Cascade Viscous Flow Analysis Using the Navier-Stokes Equations.* AIAA Paper 86-0033, 1986.

[3] J. G. Kallinderis. *Space, Time and Equation Adaptation for Viscous Flows.* Ph.D. Thesis, Massachusetts Institute of Technology, to appear Spring 1989.

[4] J. G. Kallinderis and J. R. Baron. *Adaptation Methods for a New Navier-Stokes Algorithm.* AIAA Paper 87-1167-CP, 1987(to appear in AIAA Journal Sept. 1988).

[5] J. G. Kallinderis and J. R. Baron. *Unsteady and Turbulent Flow using Adaptation Methods.* Proceedings of the 11th Int. Conf. on Numerical Methods in Fluid Dynamics, Williamsburg, Virginia, 1988.

[6] W. Kordulla. *Integration of the Navier-Stokes Equations in Finite-Volume Formulation. Computational Fluid Dynamics.* VKI Lecture Series 1987-04, Mar 1987.

[7] R. W. MacCormack. *Current Status of Numerical Solutions of the Navier-Stokes Equations.* AIAA Paper 85-0032, 1985.

[8] R. W. MacCormack and H. Lomax. Numerical solution of compressible viscous flows. *Ann. Rev. Fluid Mech.*, 11:289–316, 1979.

[9] L. Martinelli, A. Jameson, and F. Grasso. *A Multigrid Method for the Navier-Stokes Equations.* AIAA Paper 86-0208, 1986.

[10] R.-H. Ni. A Multiple Grid Scheme for Solving the Euler Equations. *AIAA Journal*, 20:1565–1571, Nov 1981.

[11] R. Peyret and T.T. Taylor. *Computational Methods for Fluid Flow.* Springer-Verlag, 1983.

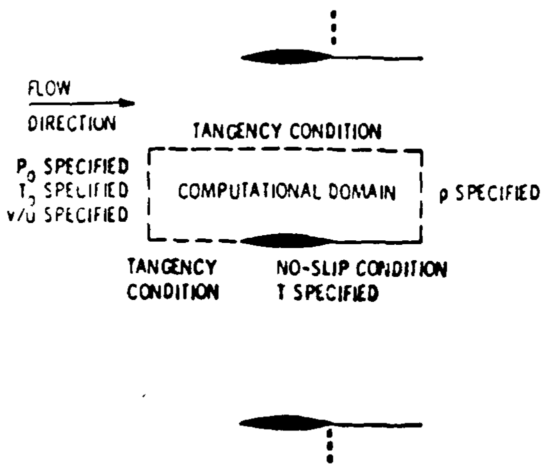
[12] R. Peyret and H. Viviand. *Computation of Viscous Compressible Flows based on the Navier-Stokes Equations.* AGARD-AG-212, 1975.

[13] R. C. Swanson and E. Turkel. *A Multistage Time-Stepping Scheme for the Navier-Stokes Equations.* AIAA Paper 85-0035, 1985.

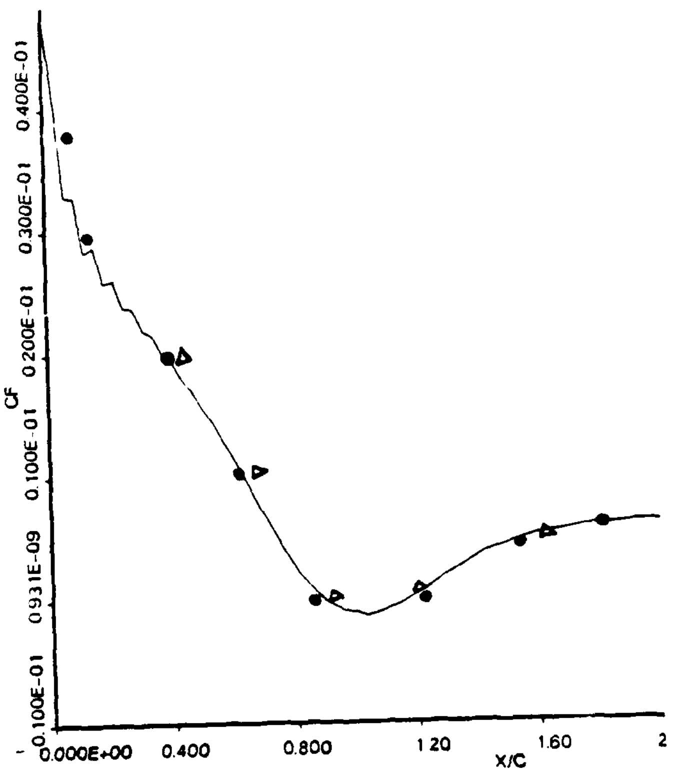
[14] E. Turkel, S. Yaniv, and U. Landau. *Accuracy of Schemes for the Euler Equations with Non-Uniform Meshes.* AIAA Paper 86-0341, 1986.

[15] M Vinokur. *An Analysis of Finite-Difference and Finite-Volume Formulations of Conservation Laws.* Technical Report NASA CR 177416, NASA, 1986.

[16] N. P. Weatherill, L. J. Johnston, A.J. Peace, and J.A. Shaw. *A Method for the Solution of the Reynolds-Averaged Navier-Stokes Equations on Triangular Grids.* Proceedings of the seventh GAMM Conference, pp 418 - 425, 1988.



(a) Computational Domain and Boundary Conditions



(b) Comparison of skin-friction distributions
– present scheme[4]
● Davis[2]
Δ Chima [1]

Figure 16: 10% circular arc cascade ($M_\infty = 0.5, Re = 8 \times 10^3$)

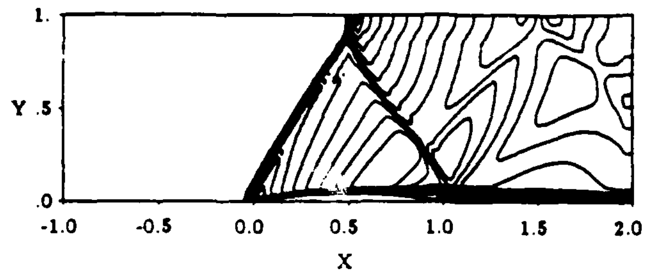


Figure 17: Mach number contours for 8% circular arc cascade ($M_\infty = 1.4$)

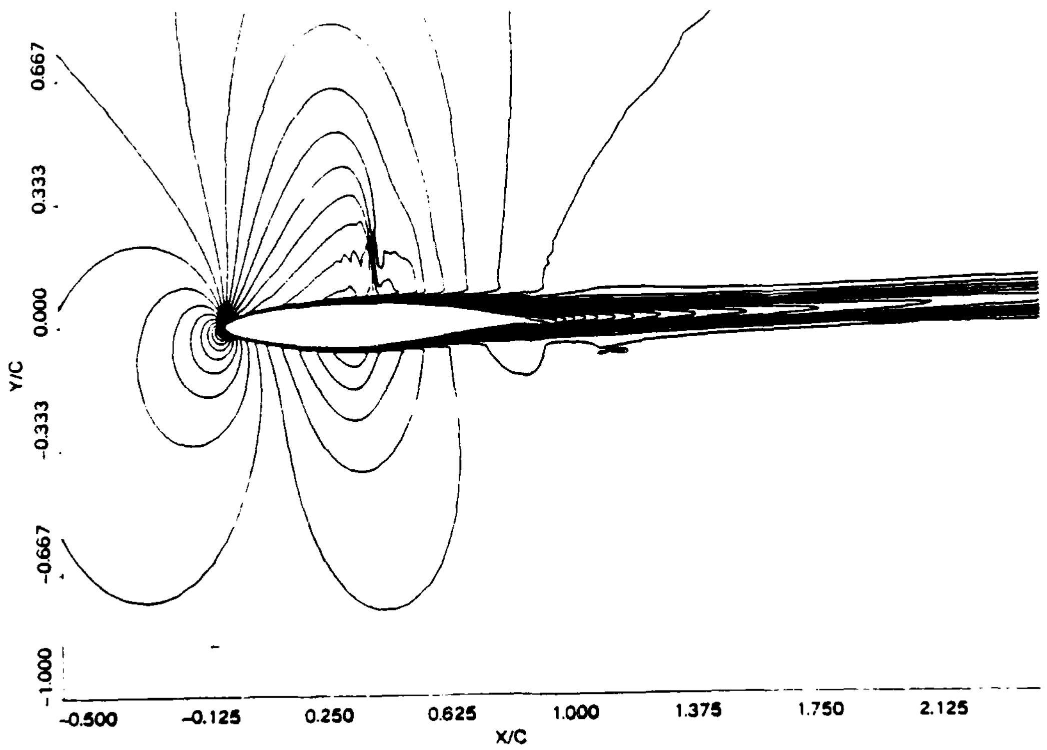


Figure 18: Mach number contours for RAE 2822 airfoil ($M_\infty = 0.75, Re = 6.2 \times 10^6$)

Application of an Adaptive Algorithm to Single and Two-Element Airfoils in Turbulent Flow

Yannis Kallinderis*

Judson R. Baron†

Computational Fluid Dynamics Laboratory

Department of Aeronautics and Astronautics

Massachusetts Institute of Technology

Cambridge, MA 02139

Abstract

An adaptive algorithm for turbulent flows, which has been developed recently, is applied to airfoil flow fields for relatively high Reynolds numbers. The adaptive algorithm employs both grid embedding and redistribution, as well as equation adaptation, in order to compute viscous flows.

Two kinds of geometries are considered. The first involves a single element NACA 0012 airfoil; the second is a two-element NLR airfoil, consisting of a main airfoil and a flap. Specifically, the NACA 0012 airfoil is considered for both subsonic and transonic flow. The flow past the two-element airfoil configuration is low subsonic and considers two flap deflection angles. The numerical results are compared with corresponding NLR experimental measurements. Important flow physics, such as shock-boundary layer interactions and small separation bubbles, are 'captured' by the adaptive algorithm with considerable detail.

INTRODUCTION

In recent years, considerable progress has been made in the development of numerical methods for the solution of the Navier-Stokes equations. Most of those methods however, are not practical for the calculation of complicated flows in a design environment. The primary reason is that the efficiency of current algorithms is poor and makes it difficult to obtain accurate results. Very fine resolution is needed, which results in long computation times even with the use of available supercomputers.

*member AIAA

†Professor, Associate Fellow AIAA

In general the selection of the equations which are to be solved, of the scheme and of the grid are determined *a priori* by the user, and quite often some or all of the above factors must be modified by the user in order to improve the results. The robustness of current numerical schemes as well as present computer capabilities have recently allowed a dramatic change in this philosophy. General algorithms have been developed which are flexible enough to adaptively adjust the equations and the grid during the solution procedure without intervention by the user [3,6,7,10], and [12,15,17].

Such an adaptive algorithm has been developed in [10], and is applied to turbulent flows around single and two-element airfoils at relatively high Reynolds numbers. First, the cases of a NACA 0012 airfoil in both subsonic and transonic flow are presented. Then, flows around a two-element NLR airfoil are considered. Experimental measurements [16,18] are employed in order to evaluate accuracy of the algorithm. Important flow physics are 'captured' by the algorithm in considerable detail.

ADAPTIVE ALGORITHM

Numerical scheme

The two-dimensional Reynolds-averaged Navier-Stokes equations are employed. An explicit, finite volume Lax-Wendroff-type numerical scheme which was developed previously by Ni [14] for the Euler equations was used for discretisation of the convective terms.

The above scheme has been extended to include viscous terms as well [11] and is conservative. In order to accelerate convergence to the steady state a multiple grid method [14] which acts only on the convective terms was used. Its function is to accelerate the propagation of waves by using coarser than basic grids. Odd-even modes were suppressed in the essentially inviscid portion of the flow with the aid of a fourth order smoothing operator, while shocks are captured using a second order Laplacian smoothing operator.

An important property of the above scheme is that all operations can be performed within each cell without the need for any external information. This is very useful in dealing with unstructured grids. A more detailed description of the integration scheme as well as investigation of its properties can be found in [10].

Adaptation

The procedure begins with an initially coarse grid which is embedded in regions with large flow-gradients (e.g. boundary-layers, shocks, wakes etc). The algorithm senses high gradient regions and automatically divides grid-cells in such regions.

The method of grid redistribution is employed together with adaptive grid embedding in order to yield a more flexible algorithm for viscous flow computations. The use of redistribution can be advantageous in those cases where the number of nodes is sufficient, and the grid rearrangement is not so severe as to result in a skewed and stretched mesh. Redistribution has been used here in order to increase grid clustering close to the surface for airfoil flows at high ($O(10^6)$) Reynolds numbers. These flows typically require the grid spacing normal to the wall to be of order 10^{-5} chord lengths for airfoil problems. As a consequence, a large number of grid embedding levels is required to decrease wall normal spacing, which can result in excessive resolution in regions away from the wall. Another important function of grid redistribution is its facility for better alignment of the grid with flow features. A choice that must be made before the adaptive procedure starts is the initial grid that will be employed. Inclusion of redistribution in the adaptive algorithm makes the procedure much more flexible and effective in accomplishing grid scale changes.

The Navier-Stokes equations apply for most flow fields of engineering interest. Frequently however, not all of its terms are necessary to model flow physics. The viscous terms in fact are expensive to compute but often are negligible over large parts of the domain. The algorithm employs the magnitude of the viscous terms as a criterion in order to decide where the full Navier-Stokes system is required and where a subset system (e.g. the Euler equations) would be adequate. The border between two such regions is dynamically defined by the algorithm and may change during the course of the solution procedure.

Turbulence model

The algebraic model due to Baldwin and Lomax [2] was used as a turbulent flow description. That model implicitly assumes a structured mesh, and its implementation is usually along lines normal to the surface. Unfortunately, for an unstructured mesh (quadrilateral or triangular), such normal mesh lines generally do not exist. Generally, interfaces interrupt such lines.

Our approach implements the model in a 'cell-wise' manner. All necessary quantities are calculated at the center of each cell. In this way we avoid using information from outside of the cell, an approach which is common when dealing with unstructured meshes generally. For example, vorticity which is an important quantity for both the inner and outer layer formulation of the model, is calculated using Green's theorem over each cell. Specifically, $\omega = -(1/S_{cell}) \oint_{cell}(udx + vdy)$ where S_{cell} is the cell area. The distance of each cell from the wall is calculated and stored whenever the grid is updated. The only quantities that require information from outside of each cell in order to be evaluated are the Baldwin-Lomax parameters *Fmax* and *Udiff* which are used for the outer layer. In order to evaluate the variables which characterise the entire shear layer profile at each streamwise location, the cells are arranged in streamwise stations. The stations consist of cells from the initial mesh plus those cells that are introduced by embedding.

APPLICATIONS TO AIRFOIL FLOWS

The adaptive algorithm is applied to airfoil flow fields for relatively high Reynolds numbers. Two kinds of geometries are considered. The first involves a single element NACA 0012 airfoil; the second is a two-element NLR airfoil, consisting of a main airfoil with a flap.

Specifically, the NACA 0012 airfoil is considered for both subsonic and transonic flow and comparisons are made with experiment. The flow past the two-element airfoil configuration is low subsonic and considers two flap deflection angles. The numerical results are compared with corresponding NLR experimental measurements for the case of 20 degrees flap deflection.

Single Airfoil Fields

Two cases of adaptive numerical results for flow around a NACA 0012 airfoil under both subsonic and transonic conditions are presented along with experimental results obtained by an AGARD group [16]. Details of the flow fields also are presented to demonstrate the range of capabilities of the algorithm. The reported CPU-times refer to an ALLIANT FX/8 computer with three processors. A speed up factor of approximately 20 compared to a microVax computer, was attained.

NACA 0012 (subsonic)

The subsonic flow conditions were: $M_\infty = 0.50, Re = 2.91 \times 10^6, \alpha = 1.77^\circ$, where both angle of attack and Mach number values are those suggested in [16] to take into account for wind tunnel wall effects.

An initial C-mesh of 33x17 points was employed, with two levels of embedding resulting in a final number of 5225 cells within the domain. The minimum grid normal spacing at the airfoil leading edge was 9×10^{-5} chord lengths, while that for the trailing edge region is 9×10^{-4}. The spacing in the streamwise direction at the leading edge region was 0.002, while the corresponding spacing at the trailing edge is 0.026. Figure 1 illustrates the embedded grid that was employed. The case took 4000 iterations to converge (reduction of residual magnitude by three orders) and required 1.8 hours. The resulting flow field is depicted in Figure 2 in terms of Mach number contour plots. The two boundary layers thicken considerably as the trailing edge is approached, but do remain attached to the surface.

The experiment [16] provided pressure distribution data, which are compared with the numerical results in Fig. 3; the comparison shows very good agreement between numerics and measurements. The computed C_L of 0.192 compares very well with the experimental value of 0.195.

NACA 0012 (transonic)

The transonic flow conditions were: $M_\infty = 0.754, Re = 3.76 \times 10^6, \alpha = 3.02^\circ$. Again the angle of attack and Mach number values are those suggested in [16] to account for wind tunnel wall effects. An initial C-mesh of 65x41 points is applied with the farfield boundary placed at 15 chord lengths away from the airfoil. Three levels of embedding are introduced by the algorithm (with the third level being directional) and results in the final grid illustrated in Fig. 4. The final number of cells within the domain is 40440. The minimum grid normal spacing at the airfoil leading edge is 2×10^{-5} chord lengths, while the spacing at the trailing edge region is 2×10^{-4}. The spacing in the streamwise direction at the leading edge region is 3×10^{-4}, while the corresponding spacing at the trailing edge is 0.004.

Adaptive redistribution of the initial mesh was applied as depicted in Fig. 5. The figure shows both the solution just before redistribution and the resulting redistributed grid. It is observed that wall clustering at the leading edge region and at the airfoil pressure side is increased. Conversely, points are moved away from the wall at and downstream of the shock-boundary layer interaction region, since the boundary layer thickens considerably and significant flow gradients exist away from the surface.

Allowance of directional grid embedding [11] at the third level resulted in a reduction of the number of cells by 17290, which represents a significant saving in both CPU-time and computer memory. The case took 5000 iterations to converge, and the consumed computing time was 8.5 hours.

Figure 6 illustrates the flow field in terms of Mach number contours. A shock forms on the suction side at 40% of the chord, with the Mach number just upstream of the shock being 1.31. The boundary layer on the suction side of the airfoil starts to thicken upstream of the shock and separates at $X = 0.82$ close to the trailing edge. On the other hand, the pressure side boundary layer is considerably thinner and remains attached to the surface. The wiggles that are observed just upstream of the shock are odd-even modes. They exist due to the low values of artificial viscosity that were used so that the solution within the viscous region does not become contaminated. Such oscillations do not induce inaccuracies in the solution since the shock location is predicted accurately.

Interesting flow physics is revealed in the view of the shock-boundary layer interaction region provided by Fig. 7. The severe adverse pressure gradient that is induced by the normal shock causes the boundary layer to thicken considerably and eventually to separate at the foot of the normal shock. A separation bubble is formed and it is captured in detail by the adaptive algorithm. The boundary layer separates at $X = 0.36$ and reattaches at $X = 0.52$.

Next consider the grids that are created by the adaptive procedure in the above studied regions. It is difficult to portray on the same plot both inviscid and viscous region grids due to the very different cell-scales. In the following plots, the boundary layer regions are not enlarged (dark regions in the figures) but the kinds of viscous grids will be described. The regions with different directionality grids can be noticed more easily by observing the borders between such grids. Fig. 8(a) focuses on the leading edge region grid. There is directional embedding with the cells being divided in their streamwise direction. The grid at the shock-boundary layer interaction region (Fig. 8(b)) follows the local flow physics in an accurate manner. The upper 'inviscid' part of the shock is 'captured' by directional embedding with cells being divided along their streamwise direction. Conversely, the boundary layer ahead of the shock is resolved with directional division of cells along the normal to the surface direction (the dark region of the plot between X=0.2 and X=0.3). As the shock is approached, significant streamwise gradients are induced in the boundary layer and now cells are divided in both directions (between X=0.3 and X=0.5). Downstream of the shock, the boundary layer cells are again divided along the normal direction only, since there are no appreciable streamwise flow gradients (between X=0.5 and X=0.7). The streamwise gradient becomes appreciable again ahead of and at separation, which results in division of the boundary layer cells in both directions (region between X=0.7 and X=1.0).

The accuracy of the procedure may be examined by comparing the experimental pressure coefficient wall distribution with the corresponding numerical result (Fig. 9). The shock location is predicted accurately although it is a little more smeared. A fourth level of embedding that would provide a more 'crisp' shock, was not allowed due to computer limitations. The agreement remains good downstream of the shock. However, as the trailing edge is approached, the boundary layer does not resist the adverse pressure gradient and separates causing the pressure distribution to tend to level out. Such trailing edge separation is not observed in the experiment. The algebraic turbulence model that is employed is believed to be largely responsible for this behaviour as has been concluded by comparative studies of different turbulence models for transonic airfoils [5,9]. The pressures on the pressure and suction sides match at the trailing edge and the somewhat lower pressure level at the suction side influences the pressure side distribution causing it to deviate slightly from the experimental results. The deviation is approximately the same over most of the pressure surface. Unfortunately, corresponding measurements for skin-friction were not performed.

Two-Element Airfoil Fields

To date, virtually all numerical results for multi-element airfoils originate from panel methods, Euler computations [1,4,13,19,20], and viscous-inviscid interaction schemes [8]. However, there are cases for which the full Navier-Stokes equations are a necessity in order to describe important flow physics. The present adaptive algorithm appears to be promising for such computations. The use of quadrilateral meshes in the past has yielded quite awkward grids, and therefore the introduction of adaptation is of some interest for better mesh topologies. Finally, the use of quadrilateral meshes, in contrast, for example, to a triangular mesh provides a test of their suitability for complex geometries computations.

The basic airfoil section is a NLR 7301 airfoil. The flap chord is $0.32c$, where c is the main airfoil chord (Fig. 10). The overlap region between the main airfoil and the flap is $0.053c$, and a gap width of $0.026c$ was considered. During the experiment only a single flap deflection angle δ of 20 degrees down had been considered and this was duplicated for numerical simulation. Lastly, the flow field in the case of an undeflected flap angle of 0° was simulated numerically, and revealed quite interesting flow physics. However, no experimental results are available for the latter case. The two-element configuration with the two different flap positions is illustrated in Fig. 10.

The flow conditions were: $M_\infty = 0.185, Re = 2.51 \times 10^6, \alpha = 6.0^\circ$. Both laminar and turbulent flow regions were observed during the experiment and the measured transition locations on the surfaces were employed by the algorithm, since a transition model has not been incorporated into the solver. The same transition locations were employed for both flap deflection angles. Specifically, the flow along the main airfoil element suction side was assumed to be turbulent downstream of $X = 0.03c$, while the corresponding location on the pressure side was taken as $X = 0.65c$. The flap pressure side flow is laminar and the flow at the suction side becomes turbulent at a distance of $0.20c$ downstream of the flap leading edge.

Flap deflection of 20 degrees

An initial H-grid of 77x103 points is employed. Two levels of embedding were used, which resulted in 50185 cells over the entire domain. Figures 11, 12 show details of the embedded grid. The generation of a grid which satisfies certain resolution and stretching requirements, and which follows the surfaces, proves to be quite difficult. However, the above resolution and stretching requirements can be relaxed to some extent when generating the initial mesh, since the adaptive algorithm will place embedded grids in regions where additional resolution is required. The minimum grid normal spacing at both leading edges is 10^{-4} chordlengths.

The algorithm took 5000 iterations to reduce the residual by three orders of magnitude and required 24 hours on the Alliant FX/8 with three processors. The free-stream Mach number is quite low, which makes computations with an explicit scheme more expensive due to the lower time-steps that are employed.

Let us consider some of the flow physics that may be observed. Following the airfoil leading edge suction peak, the boundary layer experiences adverse pressure and separates at $X = 0.027$ forming a small bubble, as it is apparent in Fig. 13. The beginning of the bubble that is predicted agrees with the measured location. It is important to note that appearance of the bubble is very sensitive to added artificial viscosity. An increase of the fourth order smoothing coefficient from $\sigma_4 = 0.0004$ (which was used in this case) to $\sigma_4 = 0.0008$ causes the bubble to disappear. The extra dissipation that was added caused the boundary layer to remain attached. The same behaviour has been observed in a lot of cases during the course of the present work.

The overlap region between the primary airfoil and flap element is of special interest. The boundary layer on the pressure side of the airfoil is very close to separating ahead of the trailing edge region, but recovers at the trailing edge region. Fig. 14 shows the flow field in terms of Mach number contours, and velocity vector plots.

Finally, the flap leading edge region is shown in Fig. 15. Observe that the stagnation point lies on the upper surface of the flap despite the free-stream flow angle of attack being 6 degrees. The flow proceeds around the leading edge from the upper side towards the lower side of the flap. Clearly, a different gap and flap deflection angle combination may provide more useful flow conditions.

The numerical results may be compared with corresponding experimental results. Figure 16 shows good agreement

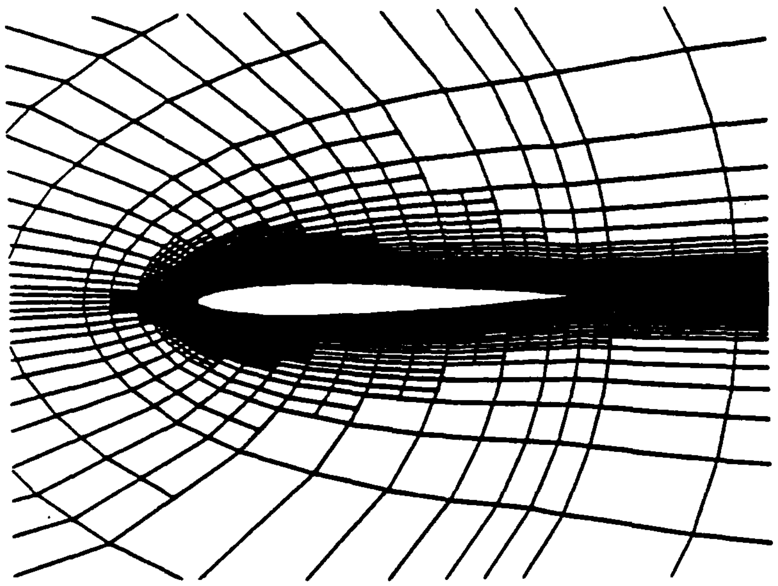


Figure 1: Two-level embedded grid - Subsonic NACA 0012 ($M_\infty = 0.5$, $Re = 2.91 \times 10^6$, $\alpha = 1.77°$) - horizontal scale enlarged

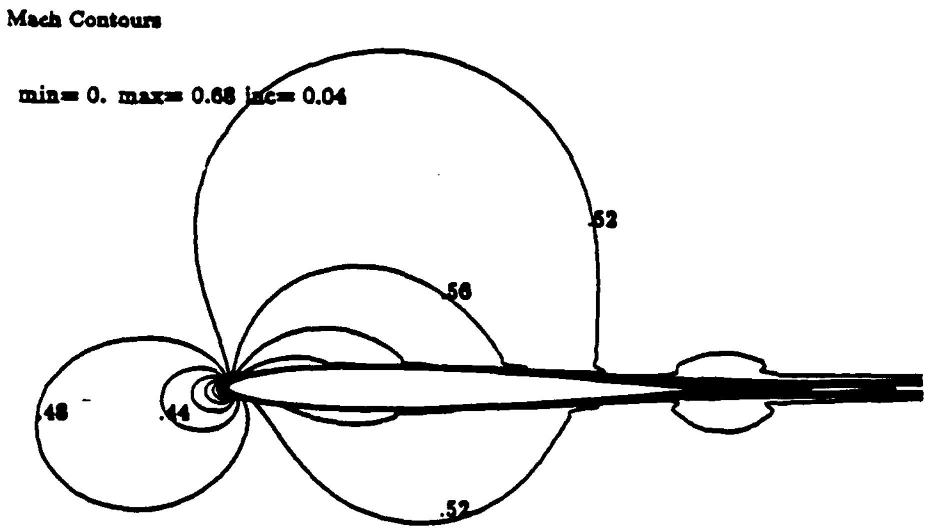


Figure 2: Flow field around a subsonic NACA 0012 ($M_\infty = 0.5$, $Re = 2.91 \times 10^6$, $\alpha = 1.77°$) - horizontal scale enlarged

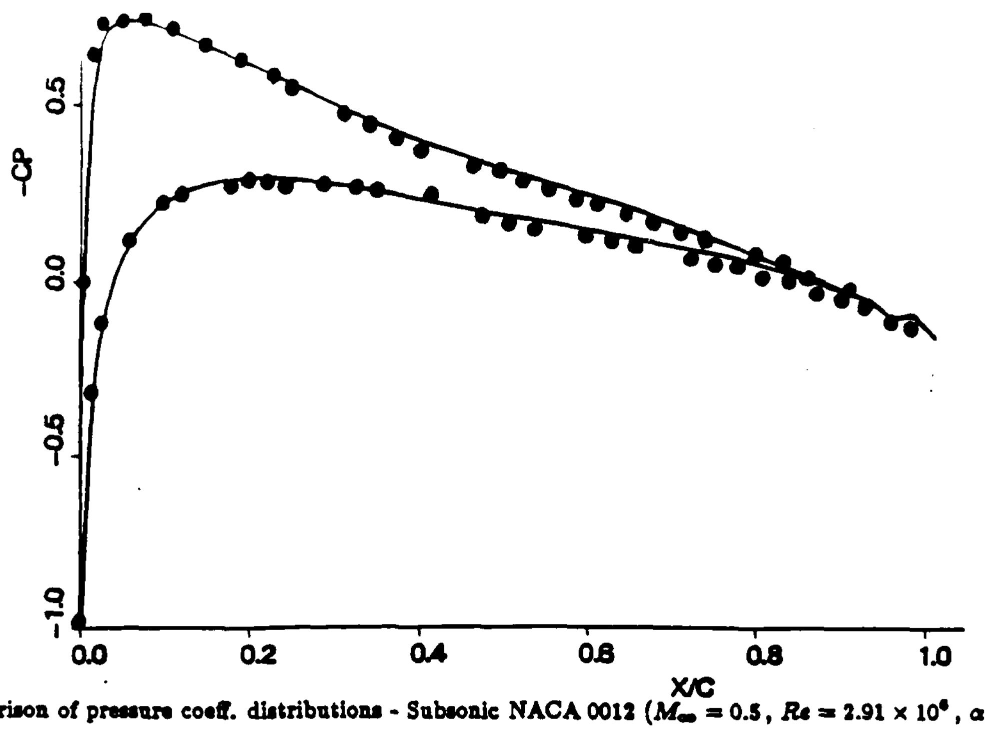


Figure 3: Comparison of pressure coeff. distributions - Subsonic NACA 0012 ($M_\infty = 0.5$, $Re = 2.91 \times 10^6$, $\alpha = 1.77°$)

between numerical and measured pressure coefficient values over both the main airfoil and the flap surfaces, with the exception of the airfoil leading edge suction peak. Higher resolution in the streamwise direction is required in order to predict the magnitude of the suction peak accurately. U-velocity profiles at various locations are compared with experimental values in Fig. 17 and believed to be the first comparisons of that kind that are available. The locations of both measurements and numerical profiles are given in the figures. Locations on both the airfoil and flap surfaces were chosen, with the profiles on the lower surfaces being inverted in the figure. Since the surface grid locations do not exactly coincide with the measurement locations, the closest surface grid location was picked for the comparisons. The comparison includes profiles in both laminar and turbulent regions. In the turbulent region, the 'laminar' sublayer is not resolved by the two-level embedded grid. A third level of adaptation is required that would lead to a prohibitively expensive computation in terms of computing time and memory requirements for the computing system that was available. Overall, the numerical results are in reasonable agreement with the measurements. It should be emphasized that in addition to the slightly different surface locations that were used, some small three-dimensional effects were reported in the experiment.

Flap deflection of 0 degrees

We proceed now to examine the previous two-element configuration but with the flap now undeflected, i.e. 20 degrees upwards relative to the previous position. It was anticipated that this 'off-design' undeflected position of the flap would 'block' the flow to some extent and most probably would cause the incoming airfoil pressure side boundary layer to separate.

The same initial mesh of 77x103 points was employed and two levels of embedding resulted in 58562 cells over the domain. The case took 5000 iterations on the final grid to reduce the residual error by two orders of magnitude, which required about 30 hours of computing time. In view of the vortex present in the overlap region and the separated flow (as will be seen shortly), there may be a question whether the flow is steady or unsteady. More iterations were not allowed due to CPU-time limitations. However, the limitation was not of overriding interest, since the purpose of the case is to illustrate the capabilities of the algorithm in capturing a variety of flow features that appear for this 'off-design' flap deflection angle.

The flow in the overlap region between the two bodies proves to be quite interesting. The boundary layer on the pressure side of the main airfoil separates, as shown in Fig. 18. A large recirculation area is formed at the airfoil trailing edge region. The center of this area lies approximately in the middle of the overlap region between the airfoil and the flap. The boundary layer reattaches just upstream of the trailing edge region. The separated shear layer impinges on the flap leading edge region and is 'divided' into two distinct layers. One portion of this separated shear layer fluid follows the flap upper surface, while the remaining portion passes around the flap leading edge and follows the lower surface. The adaptive algorithm 'captures' in detail regions with different flow orientation and with different velocity magnitudes. The shear layer, which emanates from the trailing edge lower side region, initially follows the trailing edge lower surface direction, but very quickly (within the next three grid points downstream) the velocity vectors turn to be parallel to the flap upper surface. An enlarged view of the flow at the flap leading edge region is quite informative as illustrated in Fig. 19. The part of the separated shear layer which arrives from the airfoil lower surface, and which follows the flap lower surface, separates and forms a small bubble as it is evident from the two velocity vectors plots.

SUMMARY

- An adaptive algorithm has been applied to flows of relatively high Reynolds numbers in both subsonic and transonic flow. Complex geometries were also considered and established the feasibility of using quadrilateral meshes for multi-element airfoil configurations. Comparisons with experimental results permitted an evaluation of accuracy of the algorithm.
- Important flow physics have been 'captured' by the adaptive algorithm with considerable detail. Flow phenomena such as leading edge stagnation flows, shock-boundary layer interactions, wakes, and separated boundary layers including small bubbles, were among the observed phenomena.

ACKNOWLEDGEMENTS

This work was supported by AFOSR Grants 82-0136 and 87-0218, Drs J.D. Wilson, J. McMichael, and L. Sakell serving as technical monitors.

References

[1] B. G. Arlinger. *Analysis of Two-Element High-Lift Systems in Transonic Flow.* ICAS Paper 76-13, 1976.

[2] B. S. Baldwin and H. Lomax. *Thin-Layer Approximation and Algebraic Model for Separated Turbulent Flows.* AIAA Paper 78-257, 1978.

[3] M. Berger and J. Oliger. Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *J. Comp. Phys.*, 53:484–512, 1984.

[4] D. Caughey. An Inviscid Analysis of Transonic Slatted Airfoil. *Journal of Aircraft*, 13, 1976.

[5] T. J. Coakley. *Numerical Simulation of Viscous Transonic Airfoil Flows.* AIAA Paper 87-0416, 1987.

[6] J. F. Dannenhoffer III. *Grid Adaptation for Complex Two-Dimensional Transonic Flows.* Technical Report Sc.D. Thesis , CFDL-TR-87-10, Dept. of Aeronautics & Astronautics , MIT, August 1987.

[7] R. L. Davis and J. F. Dannenhoffer III. *Adaptive Grid Embedding Navier-Stokes Technique for Cascade Flows.* AIAA Paper 89-0204, 1989.

[8] B. Grossman and G. Volpe. *The Viscous Transonic Flow Over Two-Element Airfoil Systems.* AIAA Paper 77-688, 1977.

[9] T. L. Holst. *Viscous Transonic Airfoil Workshop - Compendium of Results.* AIAA Paper 87-1460, 1987.

[10] I. Kallinderis. *Adaptation Methods for Viscous Flows.* Technical Report Ph.D Thesis, CFDL-TR-89-5, Dept. of Aeronautics & Astronautics , MIT, May 1989.

[11] J. G. Kallinderis and J. R. Baron. Adaptation Methods for a New Navier-Stokes Algorithm. *Journal of the American Institute of Aeronautics and Astronautics*, 27:37–43, 1989.

[12] R. Lohner. *The Efficient Simulation of Strongly Unsteady Flows by the Finite-Element Method.* AIAA Paper 87-0555, 1987.

[13] D. Mavriplis. *Accurate Multigrid Solution of the Euler Equations on Unstructured and Adaptive Meshes.* NASA Langley Research Center , ICASE rept. 88-40, 1988.

[14] R.-H. Ni. 'A Multiple Grid Scheme for Solving the Euler Equations'. *AIAA Journal*, 20:1565–1571, November 1982.

[15] M.M. Pervaiz. *Spatio-Temporal Adaptative Algorithm for Reacting Flows.* Technical Report Ph.D. Thesis, CFDL-TR-88-5, Dept. of Aeronautics & Astronautics , MIT, May 1988.

[16] J. J. Thibert, M. Granjacques, and L. H. Ohman. *NACA 0012 Airfoil.* AGARD AR 138, 1979.

[17] J.F. Thompson. *Review on the State of the Art of Adaptive Grids.* AIAA Paper 84-1606, 1984.

[18] B. Van der Berg. *Boundary Layer Measurements on a Two-Dimensional Wing with Flap.* Technical Report TR 79009 U, National Aerospace Laboratory NLR, the Netherlands, 1979.

[19] G. Volpe. *Prediction of the Flow Over Supercritical High-Lift Configurations by a Multigrid Algorithm.* AIAA Paper 84-1664, 1984.

[20] L. B. Wigton. *Application of MACSYMA and Sparse Matrix Technology to Multielement Airfoil Calculations.* AIAA Paper 87-1142-CP, 1987.

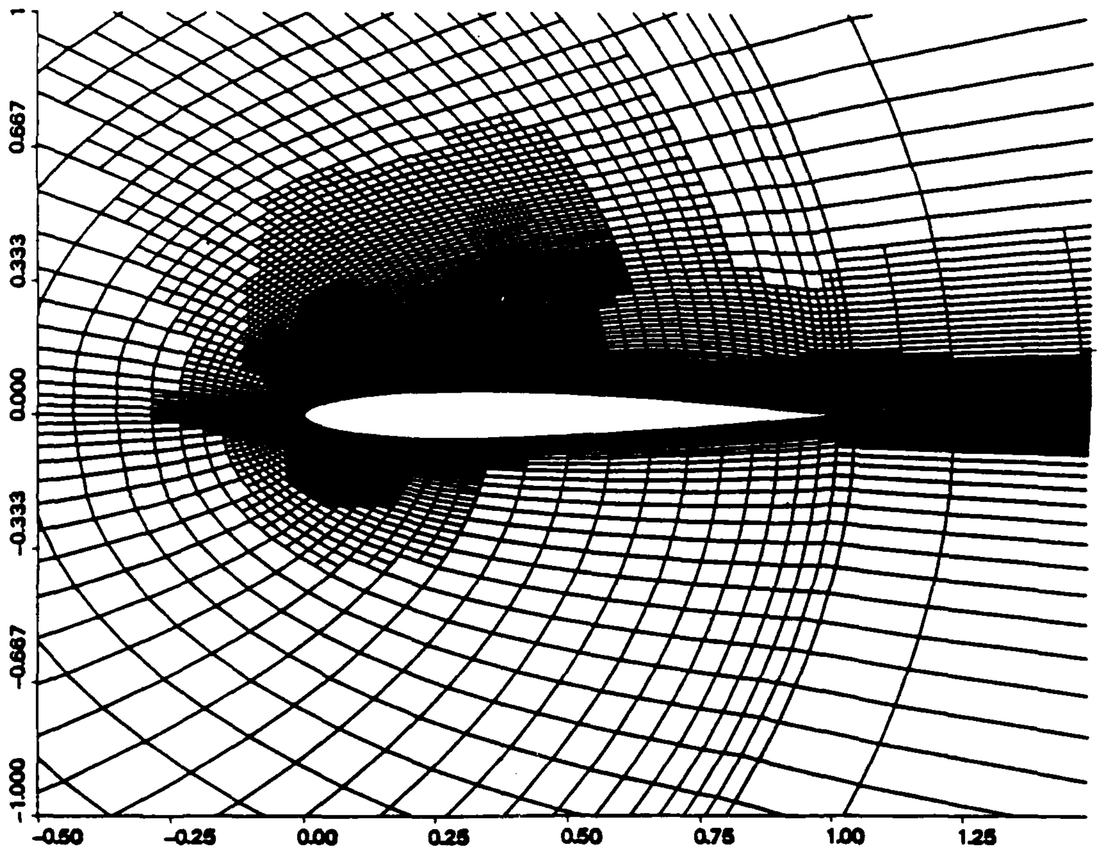
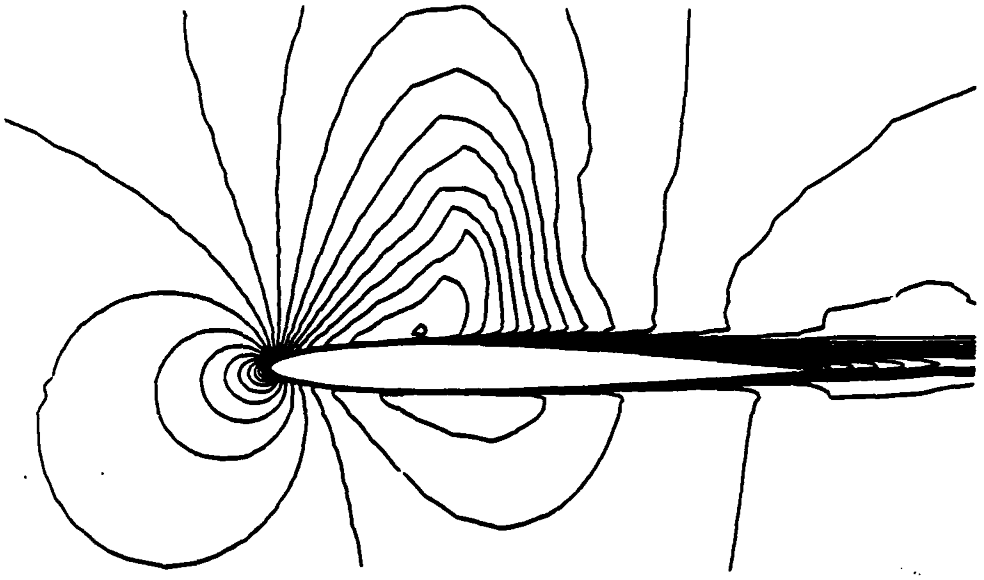
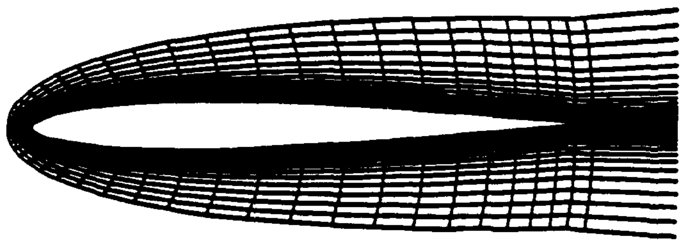


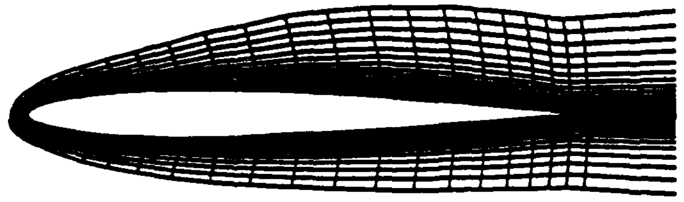
Figure 4: Final 3-level embedded grid - Transonic NACA 0012 ($M_\infty = 0.754$, $Re = 3.76 \times 10^6$, $\alpha = 3.02°$)



(a) solution before redistribution



(b) initial mesh



(c) redistributed mesh

Figure 5: Redistribution of initial grid - Transonic NACA 0012 ($M_\infty = 0.754$, $Re = 3.76 \times 10^6$, $\alpha = 3.02°$)

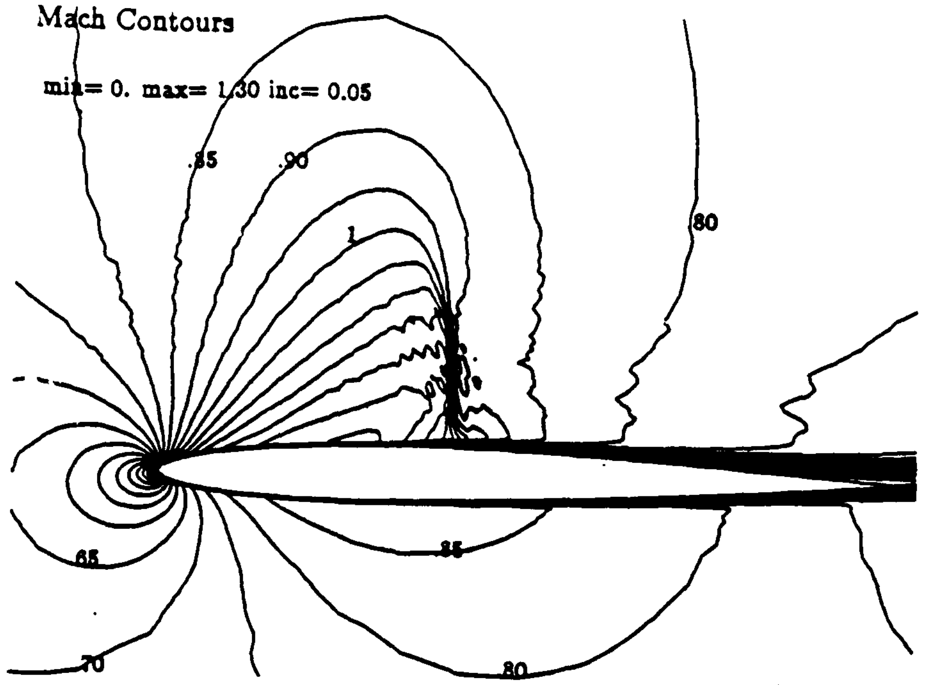


Figure 6: Flow field - Transonic NACA 0012 ($M_\infty = 0.754$, $Re = 3.76 \times 10^6$, $\alpha = 3.02°$)

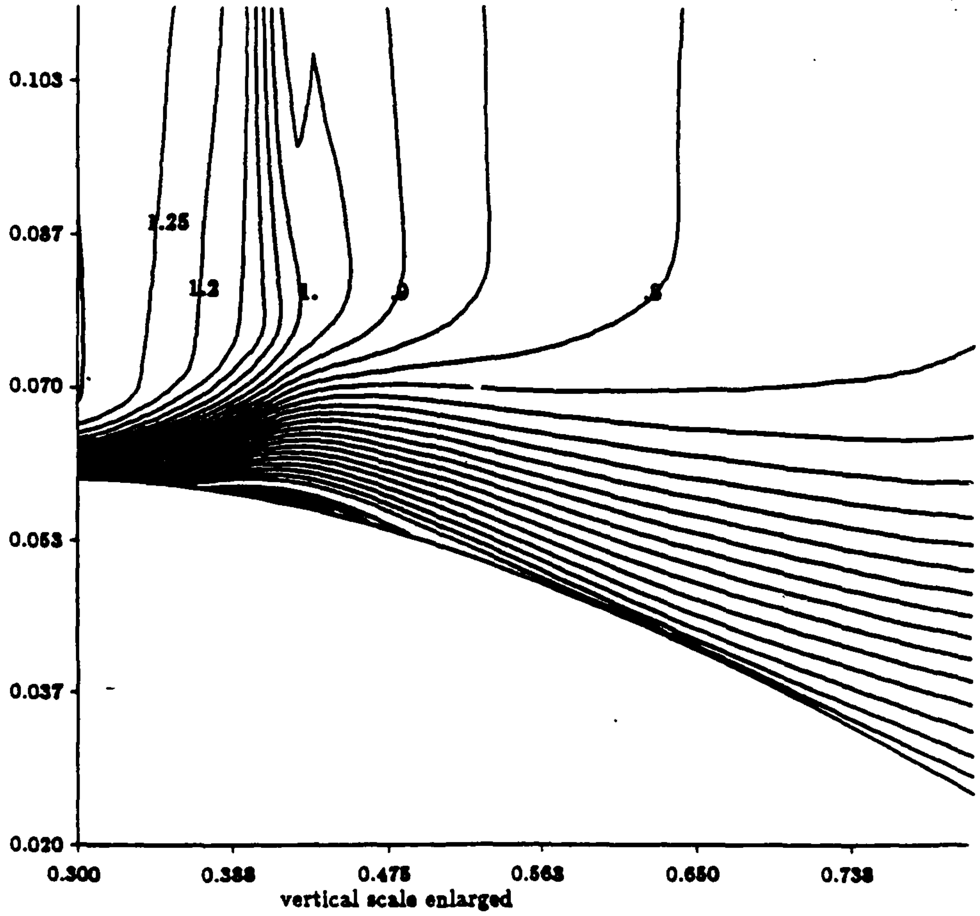
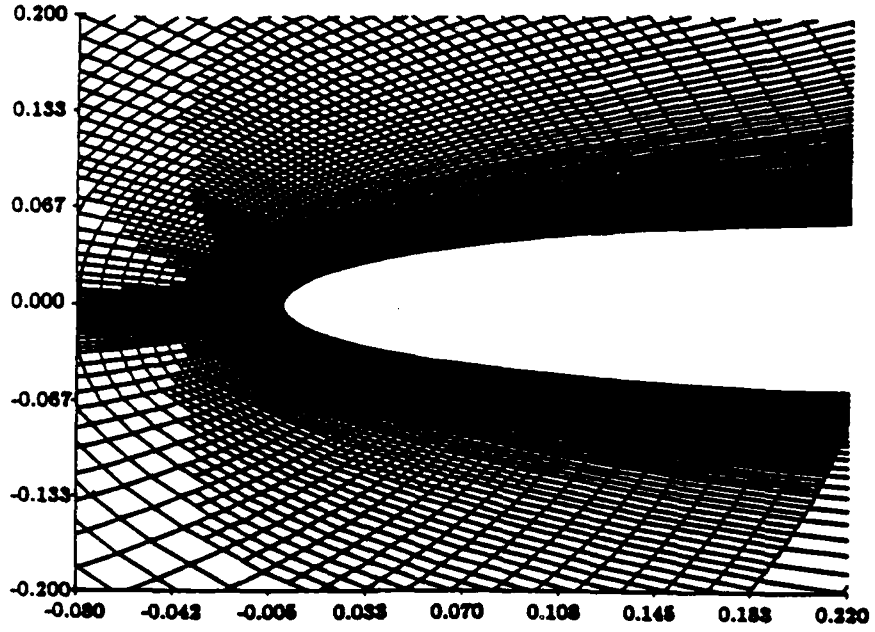
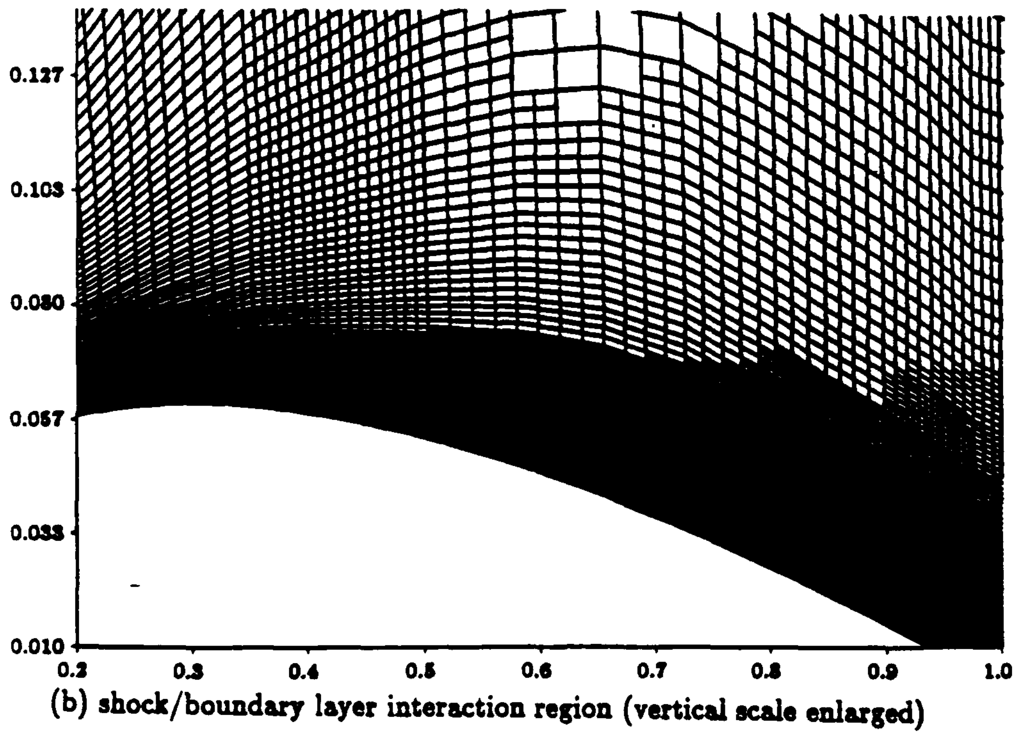


Figure 7: Separation bubble at the foot of the shock - Transonic NACA 0012 ($M_\infty = 0.754$, $Re = 3.76 \times 10^6$, $\alpha = 3.02°$)

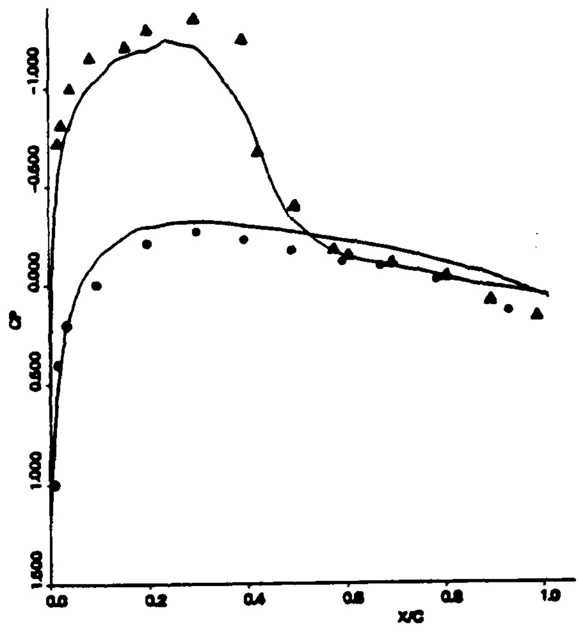


(a) leading edge region (horizontal scale enlarged)



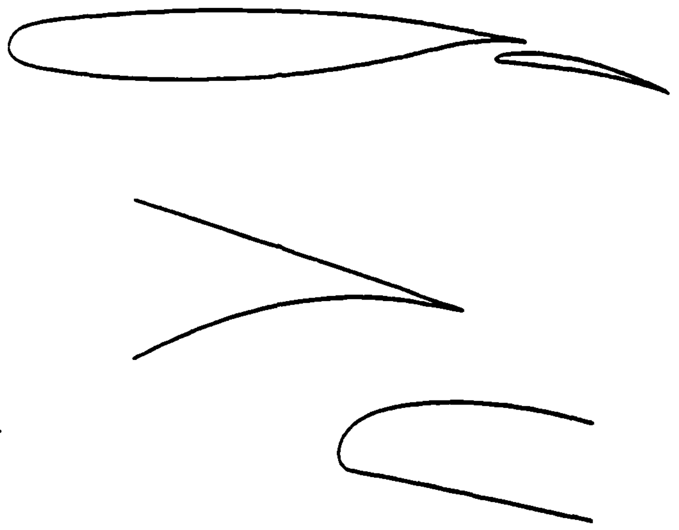
(b) shock/boundary layer interaction region (vertical scale enlarged)

Figure 8: Directional grids at various domain regions - Transonic NACA 0012 ($M_\infty = 0.754$, $Re = 3.76 \times 10^6$, $\alpha = 3.02°$)

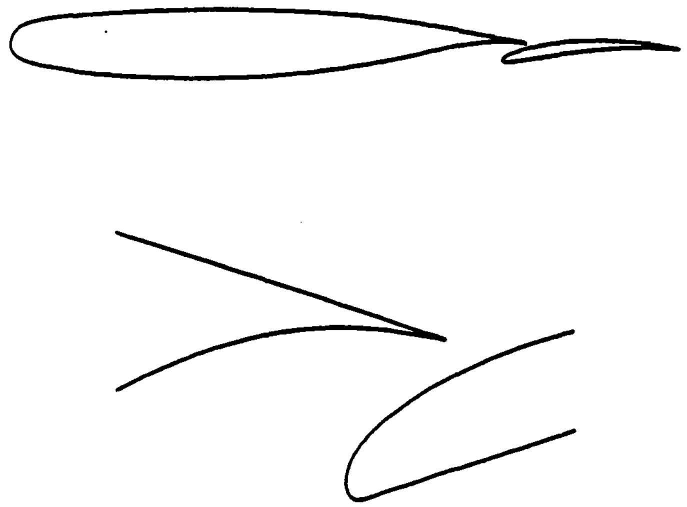


– embedded mesh , ● pressure side [16] , ▲ suction side [16]

Figure 9: Comparison of Pressure coeff. distribution with experiment - Transonic NACA 0012 ($M_\infty = 0.754$, $Re = 3.76 \times 10^6$, $\alpha = 3.02°$)



(a) flap deflection 20 deg.



(b) flap deflection 0 deg.

Figure 10: Two-Element airfoil geometry for two different flap deflection angles

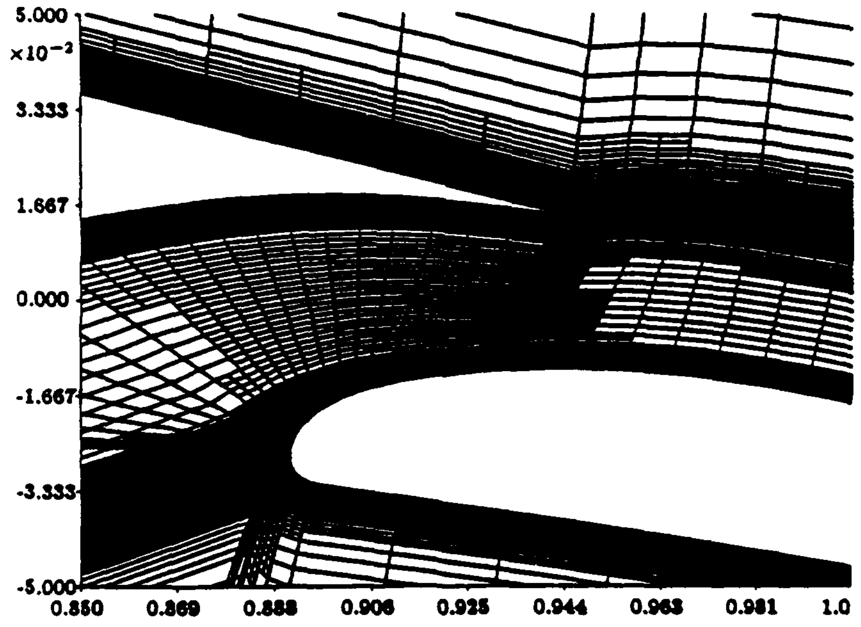
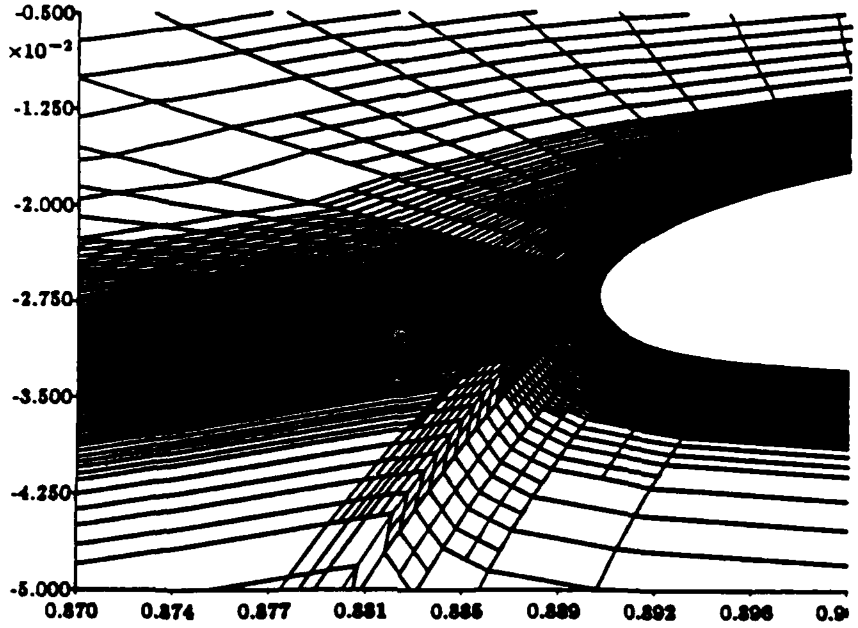
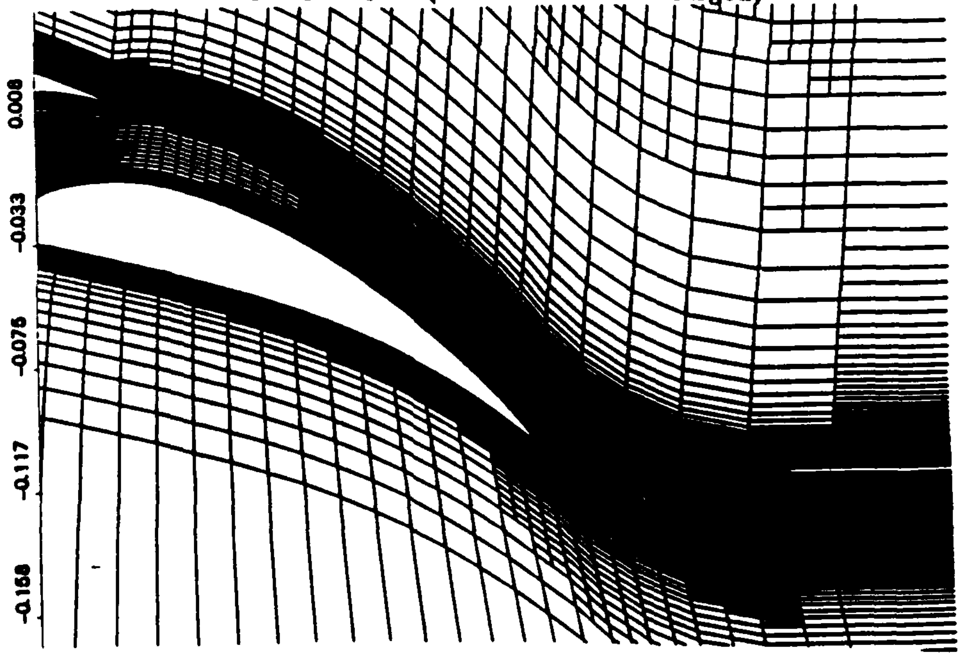


Figure 11: Two-level embedded grid - Two-Element airfoil ($M_\infty = 0.185$, $Re = 2.51 \times 10^6$, $\alpha = 6.0°$, $\delta = 20°$)

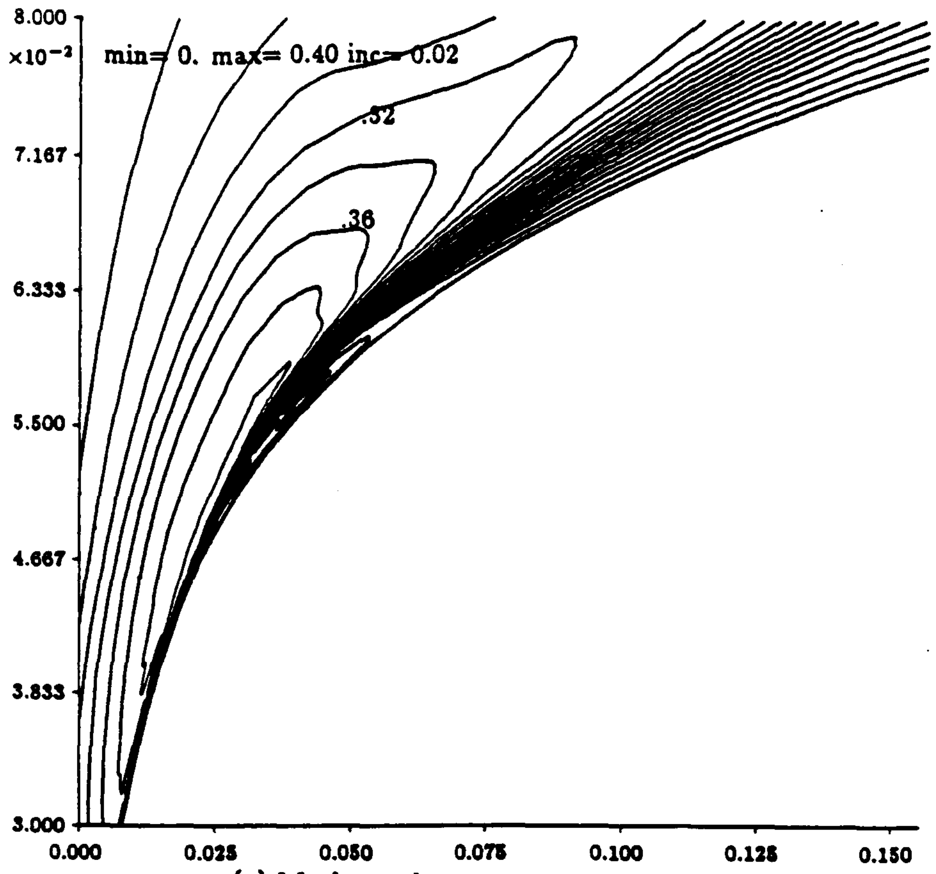


(a) flap leading edge region (horizontal scale enlarged)

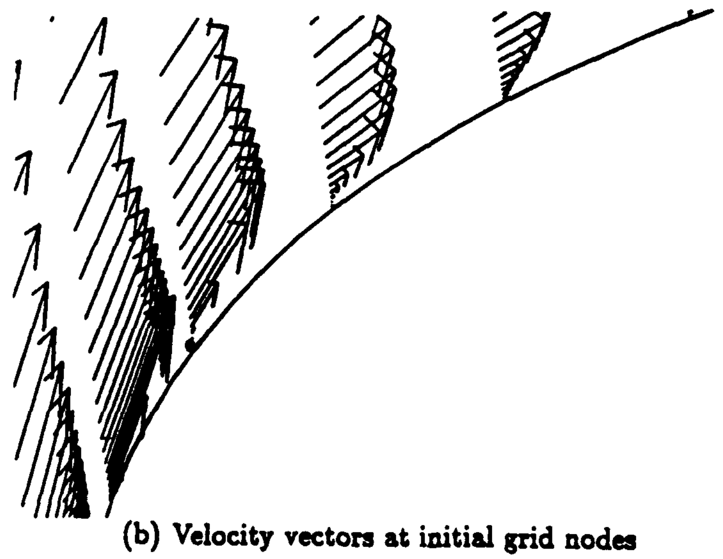


(b) flap and wakes (vertical scale enlarged)

Figure 12: Two-level embedded grid - Two-Element airfoil ($M_\infty = 0.185$, $Re = 2.51 \times 10^6$, $\alpha = 6.0°$, $\delta = 20°$)

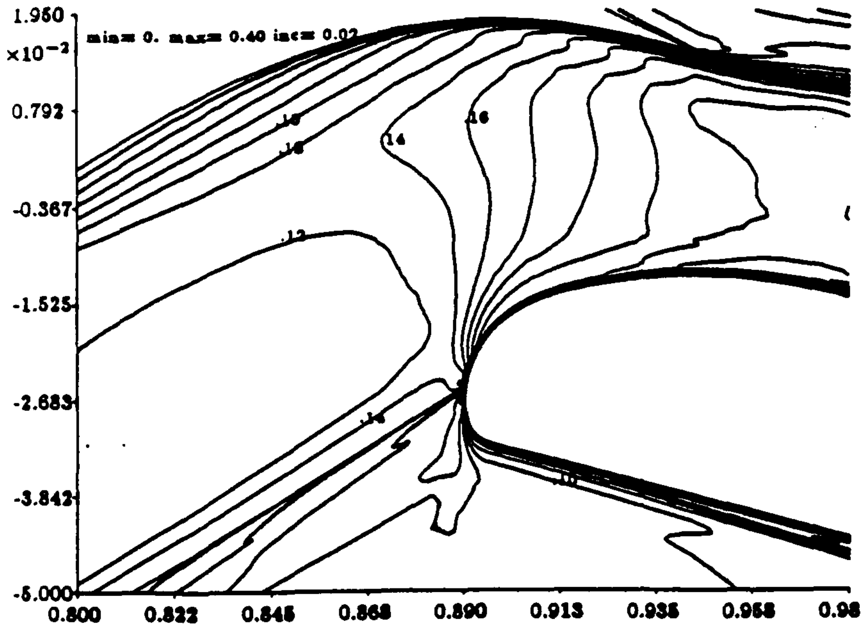


(a) Mach number contours

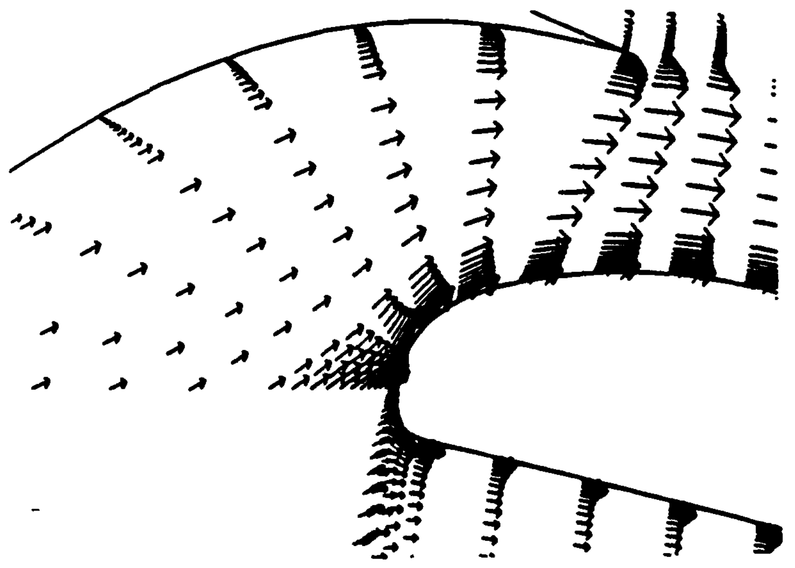


(b) Velocity vectors at initial grid nodes

Figure 13: Separation bubble at leading edge region - Two-Element airfoil ($M_\infty = 0.185$, $Re = 2.51 \times 10^6$, $\alpha = 6.0°$, $\delta = 20°$) - vertical scales enlarged

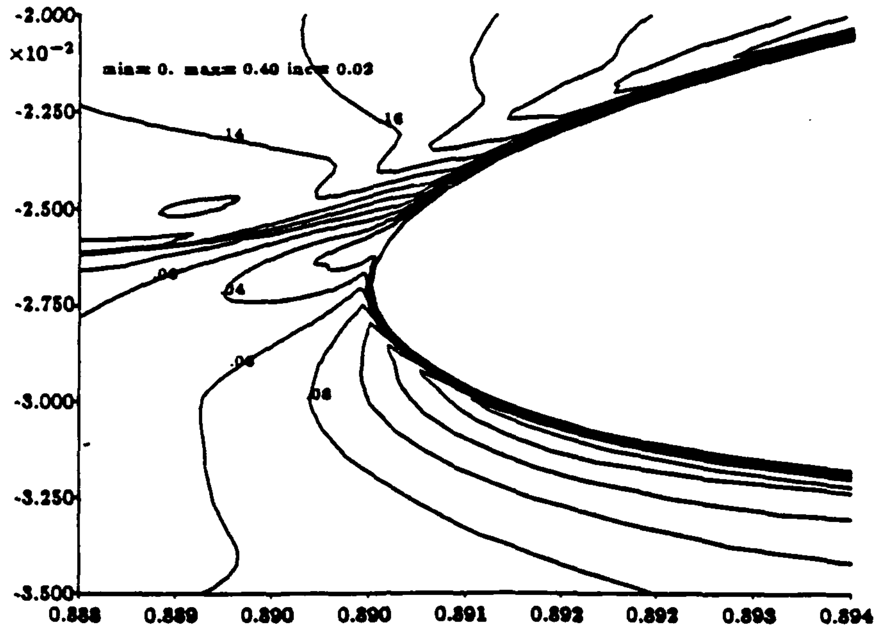


(a) Mach number contours (vertical scale enlarged)

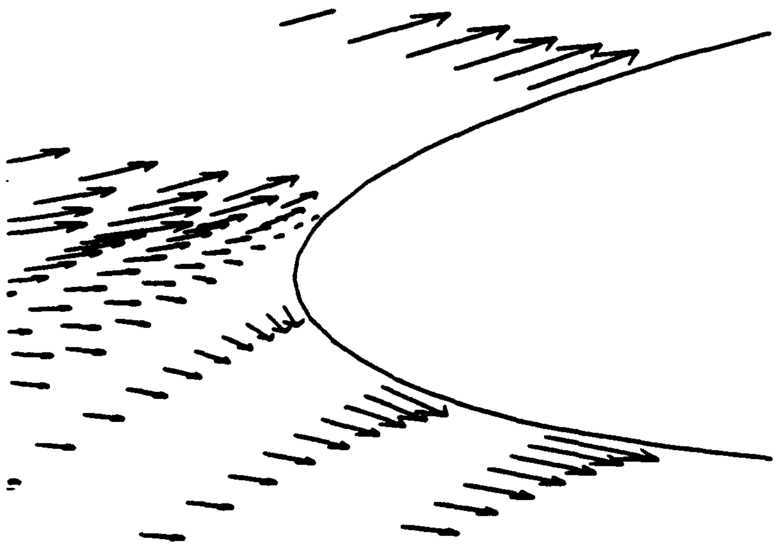


(b) velocity vectors at initial grid nodes (vertical scale enlarged)

Figure 14: Overlap region flow field - Two-Element airfoil ($M_\infty = 0.185$, $Re = 2.51 \times 10^6$, $\alpha = 6.0°$, $\delta = 20°$)

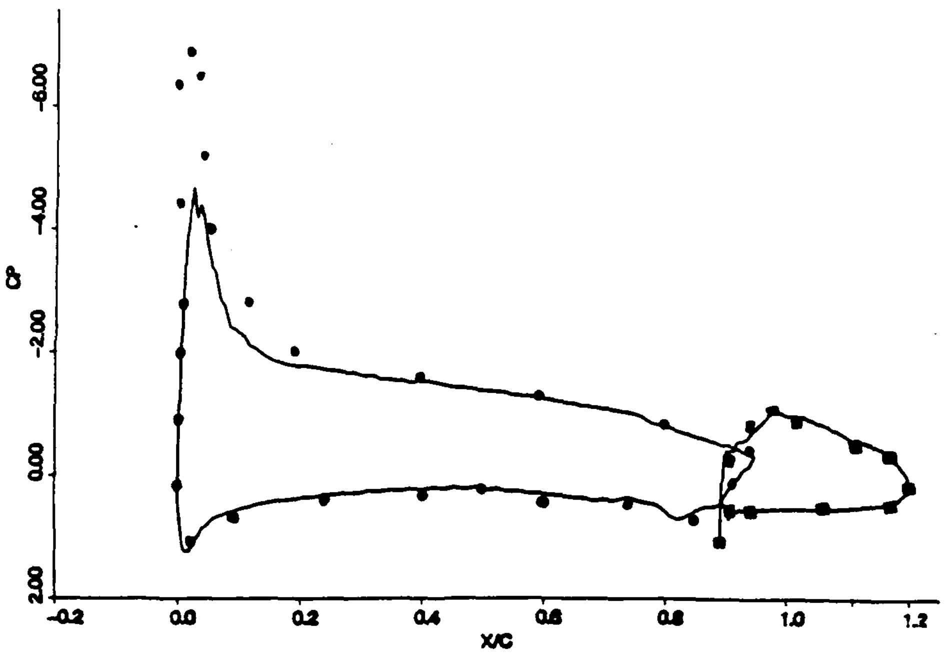


(a) Mach number contours (horizontal scale enlarged)



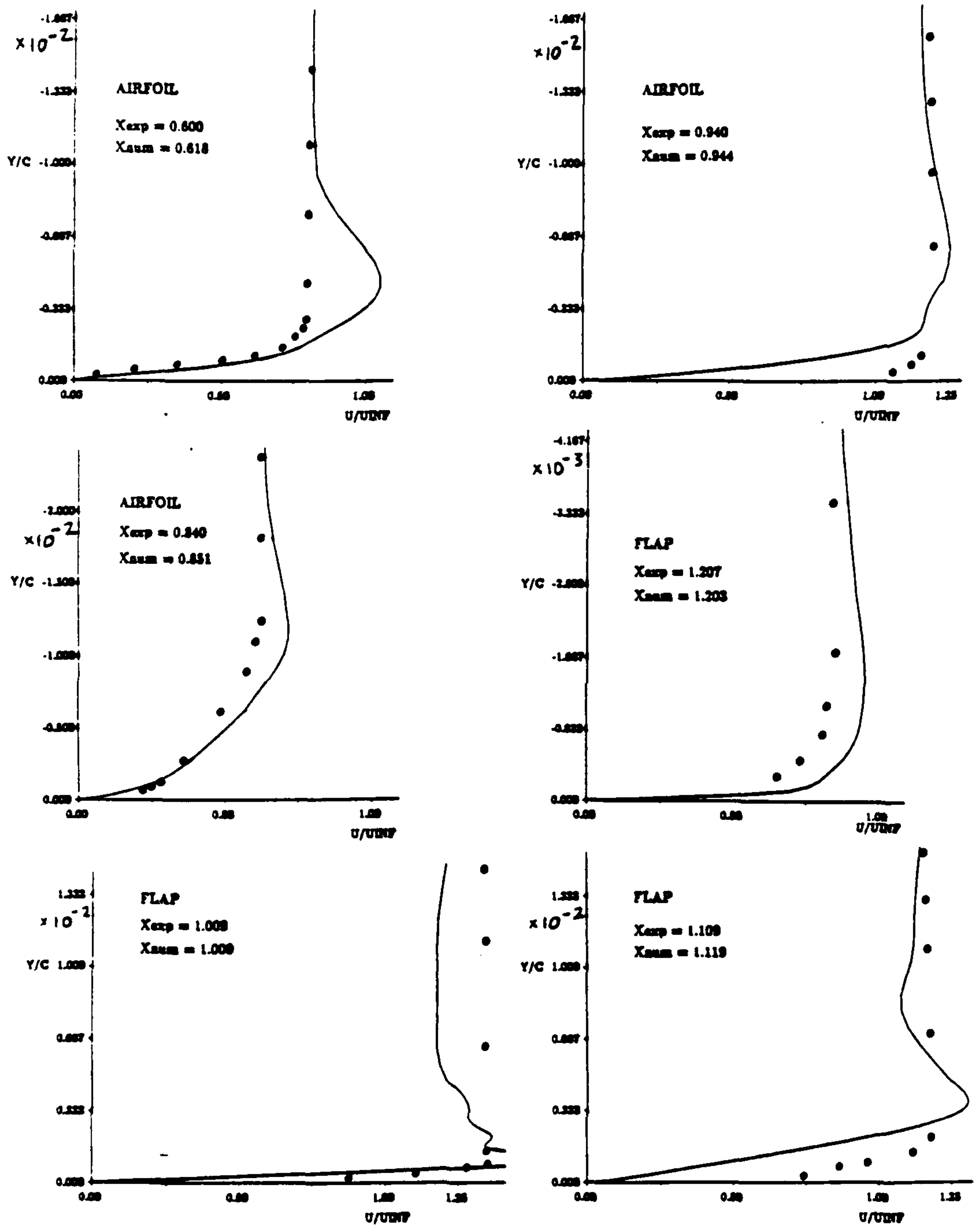
(b) velocity vectors at initial grid nodes (horizontal scale enlarged)

Figure 15: Flap leading edge region flow field - Two-Element airfoil ($M_\infty = 0.185$, $Re = 2.51 \times 10^6$, $\alpha = 6.0^\circ, \delta = 20^\circ$)



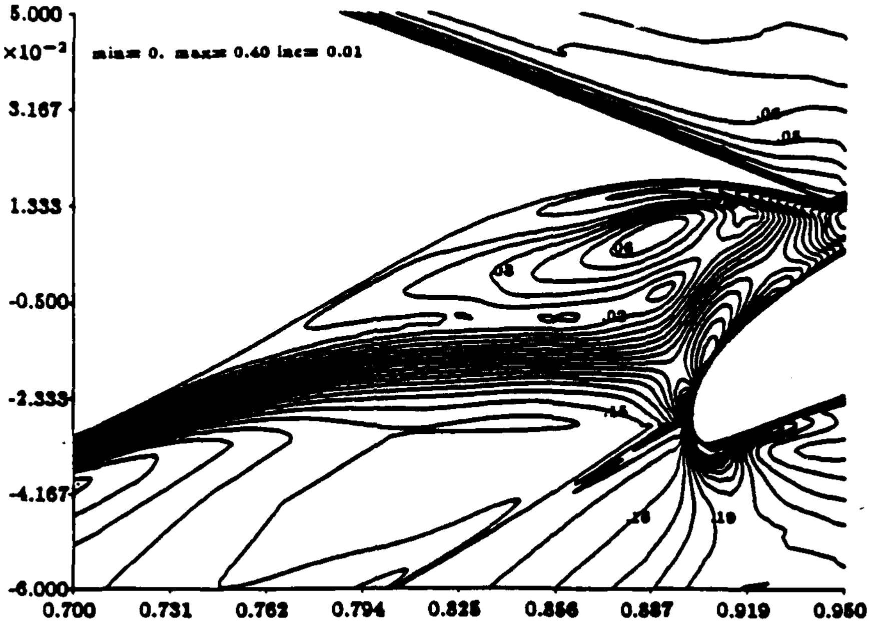
– present work , (•, ■) experiment [18]

Figure 16: Pressure coeff. comparison with experimental results - Two-Element airfoil ($M_\infty = 0.185$, $Re = 2.51 \times 10^6$, $\alpha = 6.0°$, $\delta = 20°$)

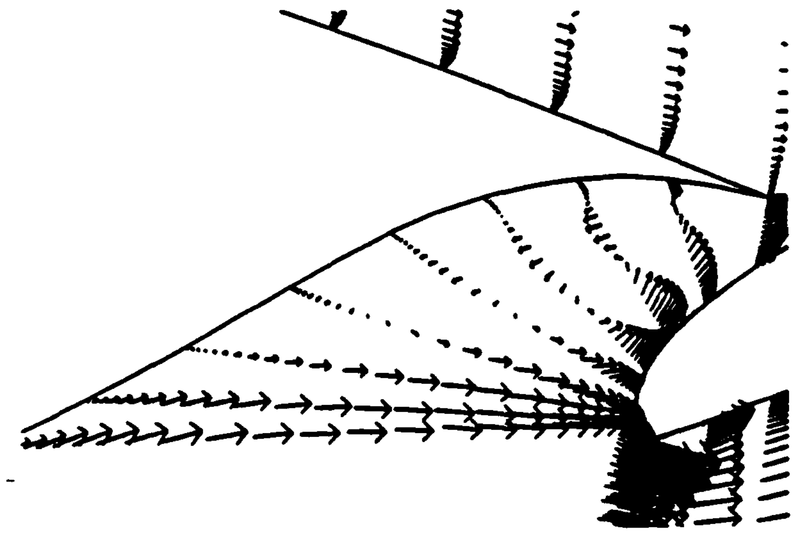


– present work , • experiment [18]

Figure 17: Comparison of U-velocity profiles with experimental results - Two-Element airfoil ($M_\infty = 0.185$, $Re = 2.51 \times 10^6$, $\alpha = 6.0°$, $\delta = 20°$)

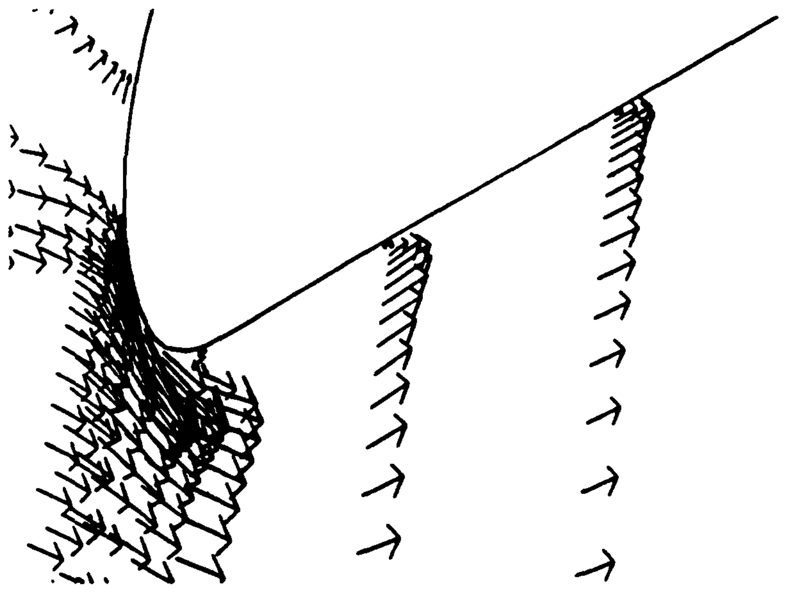


(a) Mach number contours (vertical scale enlarged)



(b) velocity vectors at initial grid nodes (vertical scale enlarged)

Figure 18: Recirculating flow between airfoil and flap - Two-Element airfoil ($M_\infty = 0.185$, $Re = 2.51 \times 10^6$, $\alpha = 6.0°$, $\delta = 0°$)



velocity vectors at initial grid nodes (enlarged view)

Figure 19: Separation bubble at flap leading edge region - Two-Element airfoil **($M_\infty = 0.185$, $Re = 2.51 \times 10^6$, $\alpha = 6.0°, \delta = 0°$)**

An Adaptive Algorithm for Turbulent Flows

Yannis Kallinderis[*] and Judson R. Baron[†]

Computational Fluid Dynamics Laboratory

Department of Aeronautics and Astronautics

Massachusetts Institute of Technology

Cambridge, MA 02139

ABSTRACT

An adaptive algorithm for simulation of laminar viscous flows, which consisted of grid embedding and equation adaptation, is extended to include turbulent fields as well as time accurate flow computations. Specifically, a combination of grid embedding and redistribution is employed for more effective grid adaptation. An algorithm that allows spatial variation of time-steps alleviates the stiffness for time accurate computations, which otherwise requires a globally minimum time-step. A method for implementing the Baldwin-Lomax algebraic turbulence model with unstructured embedded meshes is developed. A numerical treatment of grid interfaces is studied and evaluated. The adaptive algorithm is applied to airfoil flow fields at relatively high Reynolds numbers of order 10^6, and comparisons are made with experimental data.

[*]Research Assistant, member AIAA; now Assistant Professor, University of Texas at Austin

[†]Professor, Associate Fellow AIAA

[0]Copyright ©American Institute of Aeronautics and Astronautics, Inc., 1989. All rights reserved.

INTRODUCTION

Considerable progress has been made in the development of numerical methods for the solution of the Navier-Stokes equations. Most of those methods, however, are impractical for the calculation of complicated flows in a design environment. The primary reason is that the efficiency of the current algorithms is poor and makes it difficult to obtain accurate results. Very fine resolution is needed, which results in long computation times even with the use of available supercomputers.

In general the selection of the equations which are to be solved, of the scheme and of the grid are determined *a priori* by the user before starting the solution procedure, and quite often some or all of the above factors must be modified by the user in order to improve the results. The robustness of current numerical schemes as well as present computer capabilities have recently allowed a dramatic change in this philosophy. General algorithms have been developed which are flexible enough to adaptively adjust both the equations, and the grid during the solution procedure without intervention by the user. An initially coarse grid is embedded in regions with large flow-gradients (e.g boundary-layers, shocks, wakes etc). The algorithm senses high gradient regions and automatically divides grid-cells in such regions. This approach has been used for the resolution of shocks in flow fields described by Euler equations [1]. Also, an adaptive algorithm for viscous, laminar flows, which utilizes equation adaptation as well, has been developed in [2].

The method of grid redistribution can be employed together with adaptive grid embedding in order to yield a more flexible algorithm for viscous flow computations. Also, spatial variation of the time-step size may be used for time-accurate computations. All cells within an embedded zone then are integrated with the same time-step and time-steps between zones vary by factors of two. This procedure allows the use of larger than globally minimum time-steps in portions of the domain. A way of implementing the Baldwin-

Lomax algebraic turbulence model on embedded grids is proposed and investigated. A Numerical treatment of grid interfaces also is proposed and is investigated. Finally, the adaptive algorithm is applied to flows of relatively high Reynolds numbers in both subsonic and transonic flow, which involve both single and two-element airfoils. Comparisons with experiments permit an evaluation of the accuracy of the algorithm.

NUMERICAL SCHEME

The two-dimensional Reynolds-averaged Navier-Stokes equations are employed. An explicit, finite volume Lax-Wendroff-type numerical scheme which was developed previously by Ni [3] for the Euler equations is used for discretization of the convective terms.

The scheme has been extended to include viscous terms as well [2]. We illustrate the discretization by consideration of the viscous term u_{xx} at node E of Fig. 1. Using Green's theorem for the volume abcd, we have:

$$\begin{aligned} u_{xx} &= (1/S_{abcd}) \oint_{abcd} (u_x) dy \\ &= (1/S_{abcd}) \left[(u_x)_{cd} \Delta y_{cd} + (u_x)_{bc} \Delta y_{bc} + (u_x)_{ab} \Delta y_{ab} + (u_x)_{da} \Delta y_{da} \right] \end{aligned}$$

where S_{abcd} is the area of the cell $abcd$, $\Delta y_{cd} = y_c - y_d$, etc. The first order derivative at the face cd of the control volume $abcd$, is evaluated employing the area $EcFd$. Similar volumes are used for the other face derivatives. Thus,

$$(u_x)_{cd} = (1/S_{cd})[u_F \Delta y_F + u_c \Delta y_c + u_E \Delta y_E + u_d \Delta y_d]$$

where S_{cd} is the area of $EcFd$, and

$$\begin{aligned} \Delta y_F &= (\Delta y_{IF} + \Delta y_{FC})/2 \\ \Delta y_c &= (y_H + y_E)/2 - (y_I + y_F)/2 \\ u_c &= (u_E + u_H + u_I + u_F)/4 \end{aligned}$$

etc. This discretization is conservative. The discretization of the convective terms allows odd-even modes to appear in both directions but the discretization of the viscous terms eliminates such modes, and makes the scheme more robust.

In order to accelerate convergence to the steady state, a multiple grid method [3] which acts only on the convective terms was used. Its function was to accelerate the propagation of waves with the aid of coarser than basic grids. Odd-even modes were suppressed in the essentially inviscid portion of the flow with the aid of a fourth order smoothing operator, while shocks were captured using a second order Laplacian smoothing operator.

An important element of the above scheme is that all operations can be performed in a piecewise sense within each cell, without the need for any external information. This is very useful in dealing with unstructured grids. A more detailed description of the integration scheme as well as investigation of its properties can be found in [2].

COMBINATION OF GRID EMBEDDING AND REDISTRIBUTION

Accuracy is achievable with minimal additional computational effort by embedding several levels of finer grids only in those regions of the domain where important features exist.This can be accomplished simply by subdividing cells of the initial coarse grid in both cell-directions. Proceeding in that way the embedded and initial grids would be topologically similar and if the initial grid is uniform and orthogonal, these desirable properties will characterize the embedded meshes as well. However, depending on cell and feature orientation, there are situations in which resolution is needed primarily in one direction in the vicinity of the feature. In that case it is advantageous to divide the cell only in that direction and thus avoid the creation of unecessary cells (directional refinement) [2].

The earlier approach to increasing local grid resolution has been a redistribution of the existing grid points such that points are clustered in regions of interest [4]. Since the number of grid points is fixed, the clustering in one region results in less resolution in other regions. For cases in which the initial grid does not include enough points, the procedure frequently results in skewed and stretched grids which deteriorate accuracy of the scheme.

However, the use of redistribution can be advantageous when the number of nodes is sufficient, and the grid rearrangement is not so severe as to result in a skewed and stretched mesh. Redistribution has been used here in order to increase grid clustering close to the surface for airfoil flows at high [O(10^6)] Reynolds numbers. Such flows typically require grid spacing normal to the wall to be of order 10^{-5} chord lengths for airfoil problems. As a consequence, a large number of embedding levels is required to decrease the spacing, which often results in excessive resolution in regions away from the wall.

Another important function of redistribution is to provide a better alignment of the grid with flow features. A choice of initial grid must be made before the adaptive procedure starts. However, an initial mesh may be misaligned with the emerging flow features, since the solution is not known in advance. Also, it should be noted that a level of grid embedding changes grid scales only by a factor of two, and this may not be sufficient. Inclusion of redistribution as a possibility in the adaptive algorithm makes the procedure much more effective in accomplishing grid scale changes.

A redistribution algorithm has been developed to modify the initial grid adaptively and align it with the developing flow features. A measure of grid resolution in the direction normal to a surface can be values of $y^+ \equiv \frac{u_\tau y}{\nu}$, with $u_\tau = \sqrt{\frac{|\tau_w|}{\rho_w}}$ being the wall friction velocity. A criterion based on the y^+ values can be employed either to 'attract' points toward the wall, or to 'repel' them away from the surface so that a specific y^+ value is attained. The grid points are moved along the normal to the wall, and therefore the resulting grid is orthogonal to approximately the same degree as the initial mesh. In other words, this 1-D grid motion does not result in a skewed mesh. On the other hand, grid embedding introduces enough points into the domain so that excess wall clustering of points does not result in insufficient resolution elsewhere in the domain.

Specifically, the redistribution algorithm is as follows:

- integrate on the initial mesh for a number of iterations

- calculate the y^+ values at the nodes which are adjacent to the surface. Then, evaluate both the average and standard deviation for the y^+ distribution. The specified y^+-value to be attained is: $y^+_{specified} = y^+_{ave} - \alpha\, y^+_{sdev}$. The value of α was chosen to be 0.4 for this work.

- move the nodes so that $y^+ = y^+_{specified}$ adjacent to the surface. A Laplacian filter is applied to this motion in order to avoid skewed cells that would result from very different motions of neighbouring nodes. Finally, rescale all distances by which nodes will be moved so that the maximum distance is less than a certain value. This may be useful when a limited number of nodes move a very large distance compared to the rest of the nodes (e.g stagnation points, separation points).

It must be emphasized that the sole purpose of the redistribution procedure is to yield a grid which is in better alignment with the developing flow features. Choice of the coefficient α and of other parameters (essentially the value of $y^+_{specified}$) is not crucial to the procedure. The following example of a NACA 0012 airfoil in transonic flow ($M_\infty = 0.754$, $Re = 3.76 \times 10^6$, $\alpha = 3.02°$) illustrates the validity of the procedure. Figure 2(a) shows the flow field after 1000 iterations on an initial mesh of 65x41 nodes. A shock appears on the suction side of the airfoil and induces an adverse pressure gradient, which causes the boundary layer to thicken considerably. The initial grid is clustered at the surface and it is apparent that such wall-clustering is unnecessary where the boundary layer is thick. On the other hand, the leading edge region requires more resolution. Fig. 2(b),(c) show the initial and redistributed grid close to the airfoil. Comparing the two, it is observed that points have been 'attracted' at the leading edge region and at the pressure side, while nodes have been 'repelled' on that part of the suction side over which the boundary layer is thick.

TEMPORAL ADAPTATION

For explicit methods, the time step size is frequently restricted by numerical stability considerations such as the CFL condition. The time-step is proportional to cell size, and the cell dimensions for adjacent embedded zones usually differ by a factor of two. In such cases, the time steps allocated to each cell may be grouped into temporal zones. All cells corresponding to a given temporal zone then are integrated with the same time step, and the time steps between temporal zones differ by factors of two as illustrated in Figure 3. The maximum time-step in the domain is 2^n times larger than the minimum step, where n is the number of temporal zones [5,2]. Thus the cells of temporal level 1 are integrated twice using a time step δt before those of temporal level 0 that are integrated once using a time step of $2\delta t$.

This procedure allows a spatial variation of time steps while simultaneously maintaining time accuracy. In the present approach the temporal zones coincide with the embedded zones, which simplifies the algorithm considerably, compared to a method in which the temporal zones are independent from the embedding zones [6]. Figure 3 illustrates the concept with the temporal zones coinciding with the spatial ones. This splitting of the time-steps according to the embedded zones saves CPU time since not all of the cells are marched in time with the minimum time-step that is found for the entire domain. Instead, only those cells that are in the embedding zone containing the cell with the minimum time-step are integrated with that minimum time-step; the remaining cells which lie on other zones, are marched at time-steps that are multiples of the globally smallest time-step.

As a test of the time accurate procedure, the model case of a channel flow with a forced oscillation of the inlet Mach number, was considered (Figure 4). The inlet flow was varied sinusoidally according to $M_\infty = 0.8 + 0.04 \sin t$, and a low Reynolds number (10^3) was chosen in order to reduce the CPU-time required by the explicit integration basic scheme. Only one level of embedding was used as shown in the figure. Significant temporal gradients exist over the entire domain, since the solution basically follows the inlet oscillation at all grid nodes. Both curves in Fig. 4 represent time histories of the U-velocity component at a specific node of the domain. After approximately three periods, the entire flow field oscillates with the same period as the inlet. The agreement between the embedded grid and the corresponding globally fine grid is very good. Similar agreement was observed for different locations within the domain, lying in both viscous and inviscid portions of the domain. Comparing the computing times between the adapted case and the case in which the globally minimum time-step is used, temporal adaptation yields a speed-up factor of 20%.

AN ALGEBRAIC TURBULENCE MODEL WITH EMBEDDED GRIDS

The algebraic turbulence model due to Baldwin and Lomax [7] is a two layer mixing length-type model. The quantities required to evaluate the eddy viscosity can be divided into two groups: those that characterize each cell (local quantities) and others that characterize an entire boundary layer profile (global quantities). Vorticity ω and the distance y from the center of the cell to the wall, are both local cell-based quantities. The friction velocity u_τ, as well as the model parameters $y_{max}, F_{max}, u_{DIF}$ characterize an entire profile at a certain streamwise location. The latter quantities cannot be obtained in the normal way, since the presence of unstructured embedded grids requires information to be restricted to that within each cell. The turbulence model is usually implemented along lines perpendicular to the surface. However, with an embedded mesh continuous normal mesh lines do not necessarily exist because of interrupting interfaces.

The present approach applies the model in a 'cell-wise' manner, based on parameters which are known at the center of each cell. This is consistent with the overall approach when dealing with unstructured meshes; i.e. not using information from outside of a cell. The vorticity is calculated using Green's theorem over the cell: $\omega = -\frac{1}{S_{cell}} \oint_{cell} (udx + vdy)$ where S_{cell} is the cell area. The distance of each cell from the wall is calculated and stored whenever the grid is updated. In order to evaluate the variables which characterize the entire shear layer profile at each streamwise location, the cells are arranged in streamwise stations as illustrated in Fig. 5. The stations consist of cells from the initial mesh plus those cells that are introduced by the first and second embedding levels. During the higher than second level refinements, no additional stations are created and the new cells are assigned to the previously created stations.

The station quantities y_{max}, F_{max} are calculated by scanning through all cells that belong to each station. The model function F is formed for each cell, and its maximum value F_{max} which occurs at the station cell with distance from the wall equal to y_{max} is found. The wall quantity u_τ is evaluated as an average from the station cells which are adjacent to the wall [2]. Finally, the eddy viscosity values which are evaluated at the cell centers are interpolated to the nodes using a weighted interpolation. At each node, (e.g. 0 in Fig. 6) the eddy viscosity value is evaluated employing the corresponding values at the centers of the surrounding cells, according to the formula:

$$\mu_{t0} = (\frac{A - A_1}{3A})\,\mu_{t1} + (\frac{A - A_2}{3A})\,\mu_{t2} + (\frac{A - A_3}{3A})\,\mu_{t3} + (\frac{A - A_4}{3A})\,\mu_{t4}\,, \qquad (1)$$

where $A_1\,, A_2\,, A_3\,, A_4$ are surrounding cell areas, and A is their sum. The above interpolation formula reduces to linear interpolation in 1–D.

The treatment for wakes is similar with only a few modifications. The wake stations are arranged in pairs formed from upper and lower parts. The minimum velocity cell is found by scanning through the cells of both stations of each pair. Then cells 'migrate' from one station to its counterpart so that those which are above the minimum velocity cell are assigned to the upper station, and the remaining are assigned to the lower station of the pair.

The above described implementation of the algebraic turbulence model on embedded grids has been evaluated in terms of accuracy, computing time and memory requirements. The cell quantities were calculated with the locally finest cell accuracy achievable by the solver. The station quantities such as y_{max}, F_{max}, however, were calculated over cells of a station which may include cells with different streamwise locations. The essential assumption here is that no significant streamwise variation of these quantities occurs over a station. Another approximation is the interpolation of eddy viscosity values from cell-centers to nodes, which is less accurate for a stretched mesh.

A verification of the approach has been carried out for the case of subsonic flow ($M_\infty = 0.5\,, Re = 10^5$) for the 10% circular arc cascade. Transition from laminar to turbulent flow was fixed at the middle of the bump. Two levels of embedding were used as illustrated in Fig. 7(a), and comparisons of skin-friction distributions with [8] are shown in Fig. 7(b). The sudden increase in C_f at $X = 0.5$ corresponds to the transition point from laminar to turbulent flow. Additional accuracy evaluation cases, for which experimental measurements exist, are presented in a later section.

Application of the model does not require additional CPU time compared to the time which is consumed when applying the model on a regular structured mesh. The pointers which are employed are calculated only once following each embedding. Overall, the model is applied every five iterations and consumes approximately 2% of the total CPU time required by the solver.

Two additional pieces of information are necessary when applying the model on embedded meshes. These are the pointer which arranges the cells into stations, which is a 2–D array with number of elements equal to the total number of cells in the domain, and the normal distance of each cell-center from either the wall or the wake center. The additional memory that is required is about 3%.

NUMERICAL TREATMENTS OF GRID INTERFACES

Embedding of cells introduces internal boundaries (interfaces) between those cells with either different refinement levels or with different types of subdivisions. Grid interfaces in turn can be categorized into two main groups. The first are interfaces which are characterized by an abrupt change in cell size only. The grid is continuous across the interface, but cell metrics change as shown in Fig. 8(a) (*metric discontinuous grid*). The second type includes grid lines which actually are interrupted by interfaces as illustrated in Fig. 8(b) (*discontinuous grid*). In this latter case cells on the coarser side of the interface may contain additional nodes at the face midpoint.

Numerical treatments for interfaces have been examined in [9] for the potential equation, and in [10,11,12] for the Euler equations. There are a number of problems which are imposed on the integration scheme due to the presence of interfaces. The sudden change in grid size introduces a significant stretching error, which may result in a reduction of order of accuracy. Existing schemes have been developed for cells with nodes at only the four cells corners, and they require some modification in order to take into account extra face nodes. Another important issue is maintaining conservation across interfaces. The fluxes across the boundaries surrounding an interface cell should cancel one another in order for the scheme to be conservative. Other important issues are coding complexity and the ease with which an interface treatment scheme can be extended to three dimensions.

It is clear that these considerations impose serious limitations on construction of an interface scheme, and that the above requirements may easily contradict one another. In fact, simultaneous achievement of both conservation and accuracy is very difficult, and even impossible in most cases. However, not all of the above requirements are important for a specific interface. For regions in which solution variations are relatively small, clearly a reduction in order of accuracy has a negligible effect on numerical results. Conservation proves to be an important property mostly in cases of moving shocks for accurate prediction of their location and speed. However, it proves to be unimportant within a shear layer. Conversely, accuracy is more of an issue in a boundary layer, since the second order derivatives (viscous terms) are important and are more 'sensitive' to grid stretching error than are first order derivatives (inviscid terms) [2].

Consider the two types of discontinuous and metric discontinuous grids of Fig. 9 and the interface node a. For both types, cells A_1, A_2, A_3, A_4 would have normally been employed in order to evaluate the inviscid, viscous, and smoothing contributions to node a. Cells A_3, A_4 are embedded fine cells, while cells A_1, A_2 are unembedded coarse cells with vertical dimensions twice those of cells A_3, A_4. It is clear that an evaluation for example, of the viscous derivative u_{yy} at node a

$$(u_{yy})_a = \frac{u_{a1} - 2u_a + u_{a2}}{(y_{a1} - y_{a2})^2}$$

suffers from a severe stretching error. In order to alleviate the error, use is made of the 'parent' cells A_1', A_2', with roughly the same size as the coarse cells A_1, A_2. The expression for u_{yy} now becomes

$$(u_{yy})_a = \frac{u_{a1} - 2u_a + u_{a1'}}{(y_{a1} - y_{a1'})^2}$$

(Fig. 9). Accuracy is retained at the interface despite the abrupt change in cell-size. This has proven to be a very important property for cases in which the interface lies within a viscous region.

In the case of a discontinuous interface, the treatment ignores the interface face node c, and the state-variable values at the node are obtained through interpolation from corresponding values at interface nodes b,a. The above also 'ignores' the type of division (directional or two-directional) of the interface cells, and both discontinuous and metric discontinuous types of interfaces are treated identically, which simplifies the interface algorithm significantly. This is very important, if extension of the algorithm to three dimensions is of interest.

However, the above does suffer from a nonconservation error. Use of 'parent' cells A_1', A_2' instead of fine cells A_3, A_4, as well as the averaging that is used for the middle interface node c, introduces a nonconservation error. The integration would be conservative if the sum

$$\Sigma_{nodes} \frac{S}{\Delta t} \delta U \tag{2}$$

, where δU is the change in time of the state variables, S is the cell-area, and Δt is the time-step, consisted only of contributions from the boundary nodes [2]. The nonconservation error is maximum in the case of a shock which is located at and parallel to an interface. When the shock is located at coarse cell A_1, for example, the nonconservation error is of order

$$\delta_y^{A_1} P \ \delta_y^{A_1} U. \tag{3}$$

In the above expression δ_y denotes variation along one of the cell-directions. However, if the shock is located just one cell away from the interface, the error vanishes. In practice, many schemes capture a shock within approximately four cells, which implies that interfaces should be at least four cells away from shocks which are parallel to them. Shock detection can be based on monitoring $\delta_x U, \delta_y U$ which is maximum at the shock and therefore, the center of the embedded region will be at the shock.

APPLICATIONS TO AIRFOIL FLOWS

The adaptive algorithm has been applied to airfoil flow fields for relatively high Reynolds numbers. Two kinds of geometries were considered. The first involves a single element NACA 0012 airfoil; the second is a two-element NLR airfoil, consisting of a main airfoil with a flap. Specifically, the NACA 0012 airfoil is considered for both subsonic and transonic flow, while the flow past the two-element airfoil configuration is low subsonic. The numerical results are compared with corresponding experimental measurements [13,14].

NACA 0012

The subsonic flow conditions were: $M_\infty = 0.50, Re = 2.91 \times 10^6, \alpha = 1.77°$. An initial C-mesh of 33x17 points was employed, with two levels of embedding resulting in a final number of 5225 cells within the domain. The minimum grid normal spacing at the airfoil leading edge was 9×10^{-5} chord lengths, while that for the trailing edge region was 9×10^{-4}. The spacing in the streamwise direction at the leading edge region was 0.002, while the corresponding spacing at the trailing edge was 0.026. The resulting flow field is depicted in Fig. 10 in terms of Mach number contour plots.

The experiment [13] provided pressure distribution data, which are compared with the numerical results in Fig. 11; the comparison shows very good agreement between numerics and measurements. The computed C_L of 0.192 compares very well with the experimental value of 0.195.

The transonic flow conditions were: $M_\infty = 0.754, Re = 3.76 \times 10^6, \alpha = 3.02°$. An initial C-mesh of 65x41 points was applied with the farfield boundary placed at 15 chord lengths away from the airfoil. Three levels of embedding are introduced by the algorithm (with the third level being directional) and results in the final grid illustrated in Fig. 12. The final number of cells within the domain is 40440. The minimum grid normal spacing at the airfoil leading edge is 2×10^{-5} chord lengths, while the spacing at the trailing edge region is 2×10^{-4}. The spacing in the streamwise direction at the leading edge region is 3×10^{-4}, while the corresponding spacing at the trailing edge is 0.004.

The case took 5000 iterations to converge, and the computing time consumed was 8.5

hours. Figure 13 illustrates the flow field in terms of Mach number contours. A shock forms on the suction side at 40% of the chord, with the Mach number just upstream of the shock being 1.31. The boundary layer on the suction side of the airfoil starts to thicken upstream of the shock and separates at $X = 0.82$ close to the trailing edge. On the other hand, the pressure side boundary layer is considerably thinner and remains attached to the surface. The oscillations that are observed just upstream of the shock are indicative of odd-even modes. They exist due to the low values of artificial viscosity that were used to ensure that the solution within the viscous region does not become contaminated. These oscillations apparently do not induce inaccuracies into the solution since the shock location is predicted accurately.

The accuracy of the procedure also may be examined by comparing the experimental pressure coefficient wall distribution with the corresponding numerical result (Fig. 14). The shock location is predicted accurately although it is a little more smeared. A fourth level of embedding (that would provide a more 'crisp' shock), was not allowed due to computer limitations. The agreement remains good downstream of the shock. However, as the trailing edge is approached, the boundary layer does not resist the adverse pressure gradient and separates, causing the pressure distribution to tend to level out. Such trailing edge separation is not observed in the experiment. The algebraic turbulence model that is employed is believed to be largely responsible for this behaviour as has been concluded by comparative studies of different turbulence models for transonic airfoils [15,16]. The pressures on the pressure and suction sides match at the trailing edge and the somewhat lower pressure level at the suction side influences the pressure side distribution causing it to deviate slightly from the experimental results. The deviation is approximately the same over most of the pressure surface. Unfortunately, skin-friction distribution measurements were not performed.

Two-Element Airfoil

To date, the existing numerical results for multi-element airfoils include panel methods and Euler computations [17,18,19,20,21], and a viscous-inviscid interaction scheme [22]. However, the full Navier-Stokes equations sometimes are required in order to describe important flow physics. The present adaptive algorithm appears to be promising for such computations. Although the use of quadrilateral meshes in the past has yielded quite awkward grids, the introduction of adaptation promises to be of some interest for better mesh topologies. Finally, the use of quadrilateral meshes, in contrast, for example, to a triangular mesh, provides a test of their suitability for complex geometries computations.

The basic airfoil section is a NLR 7301 airfoil. The flap chord is $0.32c$, where c is the main airfoil chord (Fig. 15). The overlap region between the main airfoil and the flap is $0.053c$, and a gap width of $0.026c$ was considered. During the experiment only a single flap deflection angle δ of 20 degrees down had been considered and this was duplicated for numerical simulation. The two-element configuration is illustrated in Fig. 15.

The flow conditions were: $M_\infty = 0.185, Re = 2.51 \times 10^6, \alpha = 6.0°$. Both laminar and turbulent flow regions were observed during the experiment, and therefore the measured surface transition locations were employed by the algorithm since a transition model has not been incorporated into the solver.

An initial H-grid of 77x103 points was employed. Two levels of embedding were employed, which resulted in 50185 cells over the entire domain. The minimum grid normal spacing at both leading edges is 10^{-4} chordlengths. The algorithm took 5000 iterations to reduce the residual by three orders of magnitude and required 24 hours on an Alliant FX/8 with three processors. The free-stream Mach number is quite low, which makes computations with an explicit scheme more expensive due to the lower time-steps that are employed.

The numerical results may be compared with corresponding experimental results. Figure 16 shows good agreement between numerical and measured pressure coefficient values over both the main airfoil and the flap surfaces, with the exception of the airfoil leading edge suction peak. Higher resolution in the streamwise direction is required in order to predict the magnitude of the suction peak accurately. U-velocity profiles at various locations are compared with experimental values in Fig. 17 and believed to be the first comparisons of that kind that are available. The locations of both measurements and numerical profiles are given in the figures. Locations on both the airfoil and flap surfaces were chosen, with the profiles on the lower surfaces being inverted in the figure. Since the surface grid locations do not exactly coincide with the measurement locations, the closest surface grid location was picked for the comparisons. The comparison includes profiles in both laminar and turbulent regions. In the turbulent region, the 'laminar' sublayer is not resolved by the two-level embedded grid. A third level of adaptation is required that would lead to a prohibitively expensive computation in terms of computing time and memory requirements for the computing system that was available. Overall, the numerical results are in reasonable agreement with the measurements. It should be emphasized that in addition to the slightly different surface locations that were used, some small three-dimensional effects were reported in the experiment.

SUMMARY

- Combination of both grid embedding and grid redistribution has been applied for viscous flow adaptive computations.
- A procedure which allows a spatial variation of the time steps while simultaneously maintaining time accuracy was developed.

- An algebraic turbulence model has been applied to embedded grids in a relatively simple and accurate manner.
- An interface treatment that avoids interface grid stretching error and that is non-conservative is proposed for viscous flows that do not include moving shocks.
- The overall adaptive algorithm has been applied to flow fields which are relatively complex, as well as to flows that involve complicated geometries. Comparisons between numerical and experimental results permit evaluation of accuracy of the algorithm.

Acknowledgements

This work was supported by AFOSR Grants 82-0136 and 87-0218, Drs J.D. Wilson, J. McMichael, and L. Sakell serving as technical monitors.

References

[1] J. F. Dannenhoffer III and J. R. Baron. *Grid Adaptation for the 2-D Euler Equations.* AIAA Paper 85-0484, 1985.

[2] I. Kallinderis. *Adaptation Methods for Viscous Flows.* Technical Report CFDL-TR-89-5, Ph.D Thesis, Dept. of Aeronautics & Astronautics , MIT, May 1989.

[3] R.-H. Ni. 'A Multiple Grid Scheme for Solving the Euler Equations'. *AIAA Journal*, 20:1565–1571, November 1982.

[4] J.F. Thompson, Z. Warsi, and C.W. Mastin. *Numerical Grid Generation.* Elsevier Science Publishing Co., Inc., 1985.

[5] J. G. Kallinderis and J. R. Baron. *Unsteady and Turbulent Flow using Adaptation Methods.* Proceedings of the 11th Int. Conf. on Numerical Methods in Fluid Dynamics, Williamsburg, Virginia, 1988.

[6] M.M. Pervaiz and J. R. Baron. Temporal and Spatial Adaptive Algorithm for Reacting Flows. *Comm. in Applied Numerical Methods*, 4(1):97–111, January 1988.

[7] B. S. Baldwin and H. Lomax. *Thin-Layer Approximation and Algebraic Model for Separated Turbulent Flows.* AIAA Paper 78-257, 1978.

[8] R. L. Davis, Ron-Ho Ni, and J.E. Carter. *Cascade Viscous Flow Analysis Using the Navier-Stokes Equations.* AIAA Paper 86-0033, 1986.

[9] E. H. Atta and J. Vadyak. *A Grid Interfacing Algorithm for 3-D Transonic Flow about Aircraft Configurations.* AIAA Paper 82-1017, 1982.

[10] S.R. Allmaras and J. R. Baron. *Embedded Mesh Solutions of the 2-D Euler Equations: Evaluation of Interface Formulations.* AIAA Paper 86-0509, 1986.

[11] M. Berger. *On Conservation at Grid Interfaces.* NASA Langley Research Center , ICASE rept. 84-43, 1984.

[12] M. M. Rai. *A Conservative Treatment of Zonal Boundaries for Euler Equation Calculations.* AIAA Paper 84-0164, 1984.

[13] J. J. Thibert, M. Granjacques, and L. H. Ohman. *NACA 0012 Airfoil.* AGARD AR 138, 1979.

[14] B. Van der Berg. *Boundary Layer Measurements on a Two-Dimensional Wing with Flap.* Technical Report TR 79009 U, National Aerospace Laboratory NLR, the Netherlands, 1979.

[15] T. J. Coakley. *Numerical Simulation of Viscous Transonic Airfoil Flows.* AIAA Paper 87-0416, 1987.

[16] T. L. Holst. *Viscous Transonic A.rfoil Workshop - Compendium of Results.* AIAA Paper 87-1460, 1987.

[17] B. G. Arlinger. *Analysis of Two-Element High-Lift Systems in Transonic Flow.* ICAS Paper 76-13, 1976.

[18] D. Caughey. An Inviscid Analysis of Transonic Slatted Airfoil. *Journal of Aircraft,* 13, 1976.

[19] D. Mavriplis. *Accurate Multigrid Solution of the Euler Equations on Unstructured and Adaptive Meshes.* NASA Langley Research Center , ICASE rept. 88-40, 1988.

[20] G. Volpe. *Prediction of the Flow Over Supercritical High-Lift Configurations by a Multigrid Algorithm.* AIAA Paper 84-1664, 1984.

[21] L. B. Wigton. *Application of MACSYMA and Sparse Matrix Technology to Multielement Airfoil Calculations.* AIAA Paper 87-1142-CP, 1987.

[22] B. Grossman and G. Volpe. *The Viscous Transonic Flow Over Two-Element Airfoil Systems*. AIAA Paper 77-688, 1977.

FIGURES LABELS

Figure 1 : Integration cell

Figure 2(a) : flow field

Figure 2(b) : portion of mesh before redistribution

Figure 2(c) : portion of mesh after redistribution

Figure 2 : Initial grid redistribution - Transonic NACA 0012 ($M_\infty = 0.754$, $Re = 3.76 \times 10^6$, $\alpha = 3.02°$) - vertical scales enlarged

Figure 3 : Spatially varying time-steps

Figure 4(a) : embedded mesh

Figure 4(b) : velocity history at X = 0.5 , Y = 0.6

Figure 4 : Time accuracy for oscillating inlet flow

Figure 5 : Cells grouped by stations

Figure 6 : Interpolation of μ_t values from cell-centers to node 0

Figure 7(a) : two-level embedded mesh

Figure 7(b) : comparison of wall-shear distributions: – 2-level embedded mesh , • [8]

Figure 7 : Turbulent flow model case - 10% circular arc cascade ($M_\infty = 0.5$, $Re = 10^5$)

Figure 8(a) : metric discontinuous

Figure 8(b) : discontinuous

Figure 8 : Types of grid interfaces

Figure 9 : Interface treatment without stretching error

Figure 10 : Flow field around a subsonic NACA 0012 ($M_\infty = 0.5$, $Re = 2.91 \times 10^6$, $\alpha = 1.77°$) - horizontal scale enlarged

Figure 11 : Comparison of pressure coeff. distributions - Subsonic NACA 0012 ($M_\infty = 0.5$, $Re = 2.91 \times 10^6$, $\alpha = 1.77°$)
– present work , • experiment [13]

Figure 12 : Final 3-level embedded grid - Transonic NACA 0012 ($M_\infty = 0.754$, $Re = 3.76 \times 10^6$, $\alpha = 3.02°$)

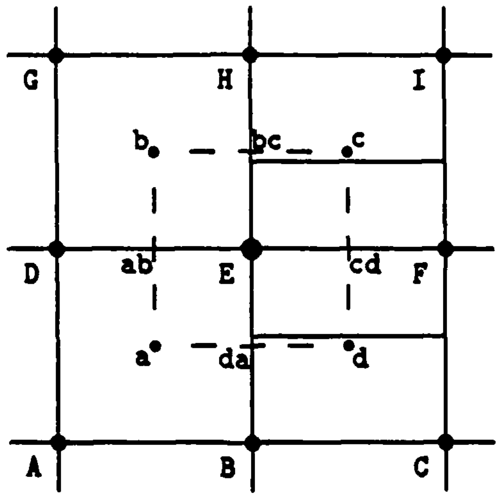
Figure 13 : Flow field around transonic NACA 0012 ($M_\infty = 0.754$, $Re = 3.76 \times 10^6$, $\alpha = 3.02°$)

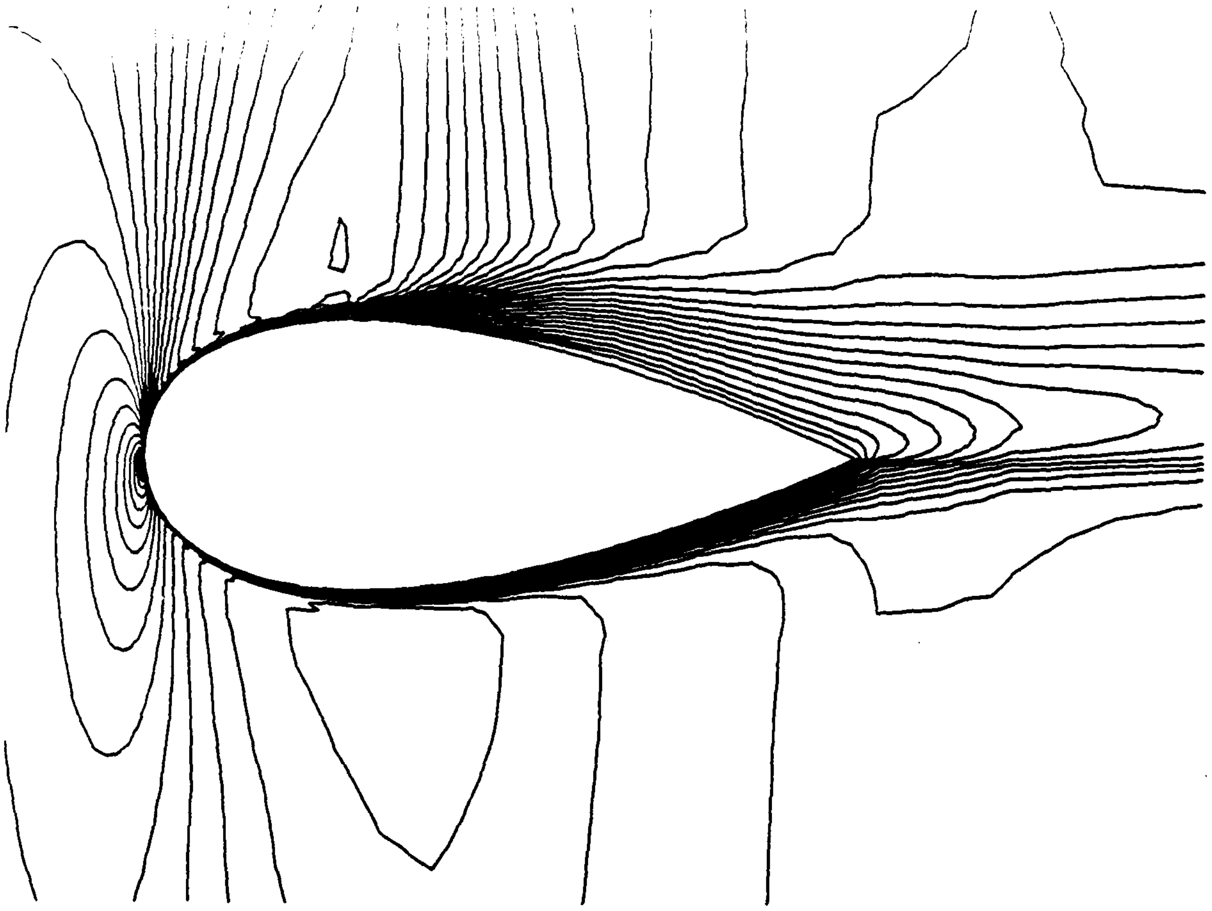
Figure 14 : Comparison of Pressure coeff. distribution with experiment - Transonic NACA 0012 ($M_\infty = 0.754$, $Re = 3.76 \times 10^6$, $\alpha = 3.02°$)
– embedded mesh , • pressure side [13] , ▲ suction side [13]

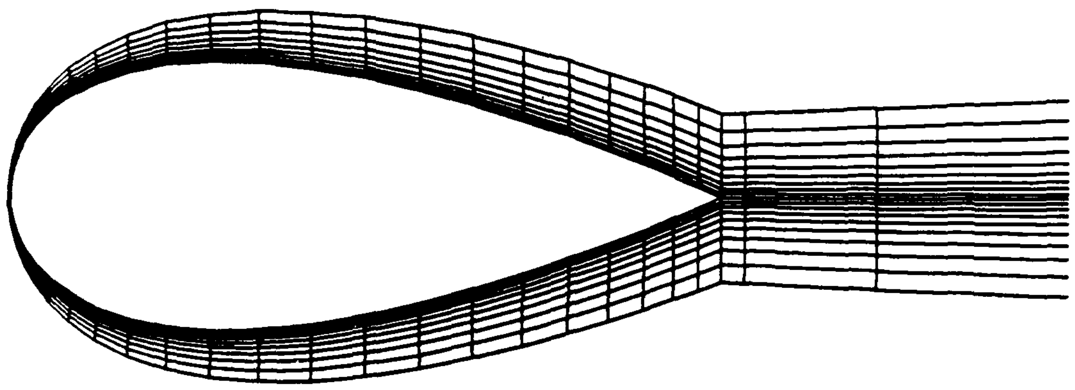
Figure 15 : Two-Element airfoil geometry

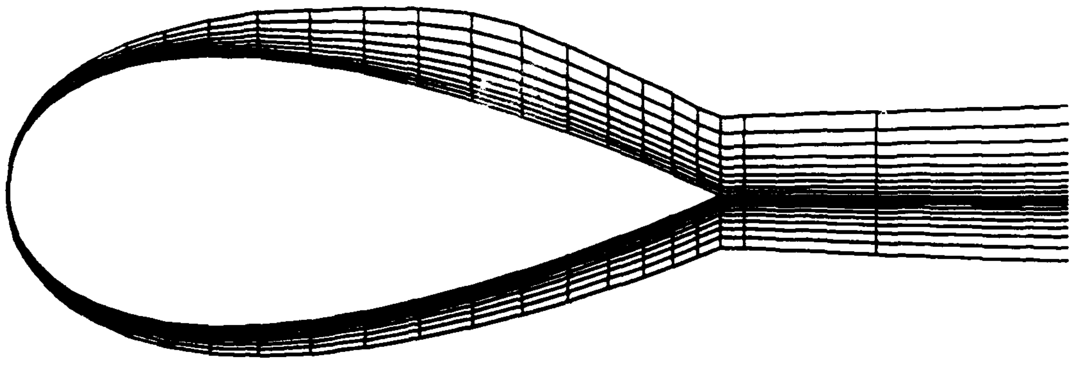
Figure 16 : Pressure coeff. comparison with experimental results - Two-Element airfoil ($M_\infty = 0.185$, $Re = 2.51 \times 10^6$, $\alpha = 6.0°, \delta = 20°$)
– present work , (•, ■) experiment [14]

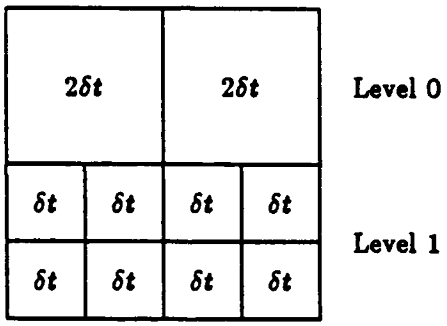
Figure 17 : Comparison of U-velocity profiles with experimental results - Two-Element airfoil ($M_\infty = 0.185$, $Re = 2.51 \times 10^6$, $\alpha = 6.0°, \delta = 20°$) – present work , ● experiment [14]

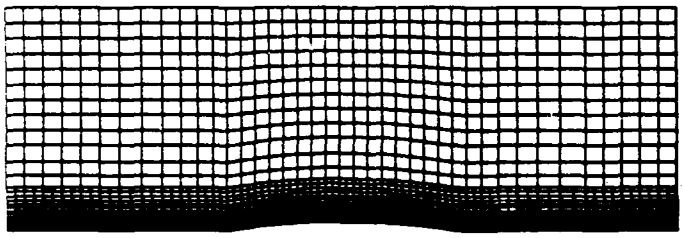

G
H
I
b
bc
c
D
ab
E
cd
F
a
da
d
A
B
C

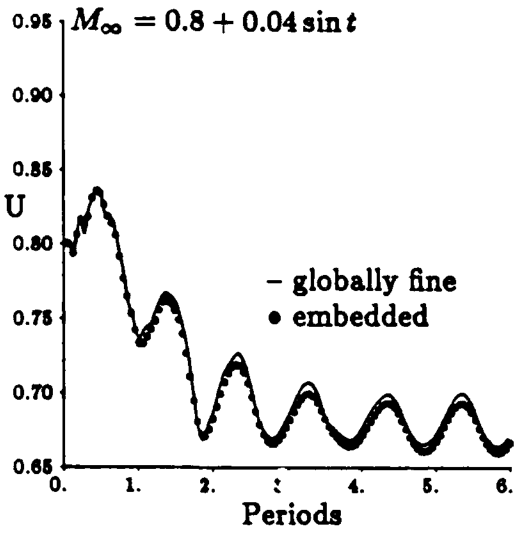




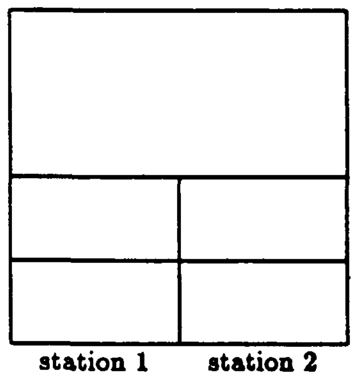


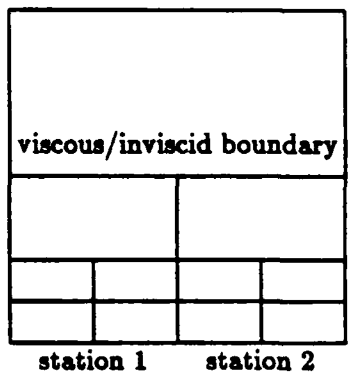

$2\delta t$
$2\delta t$
Level 0
δt
Level 1

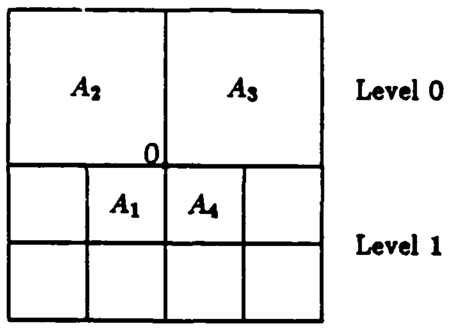


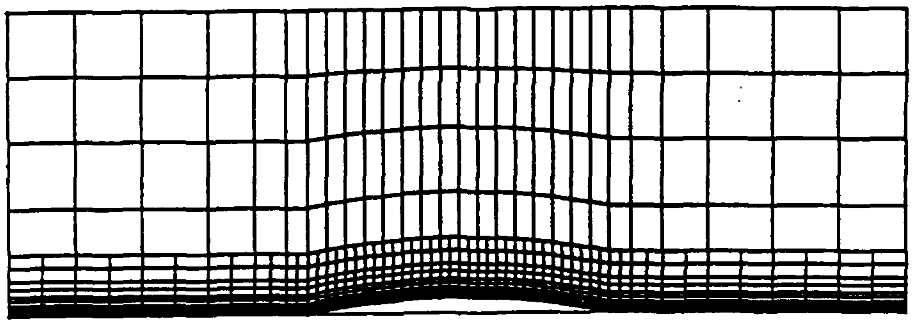


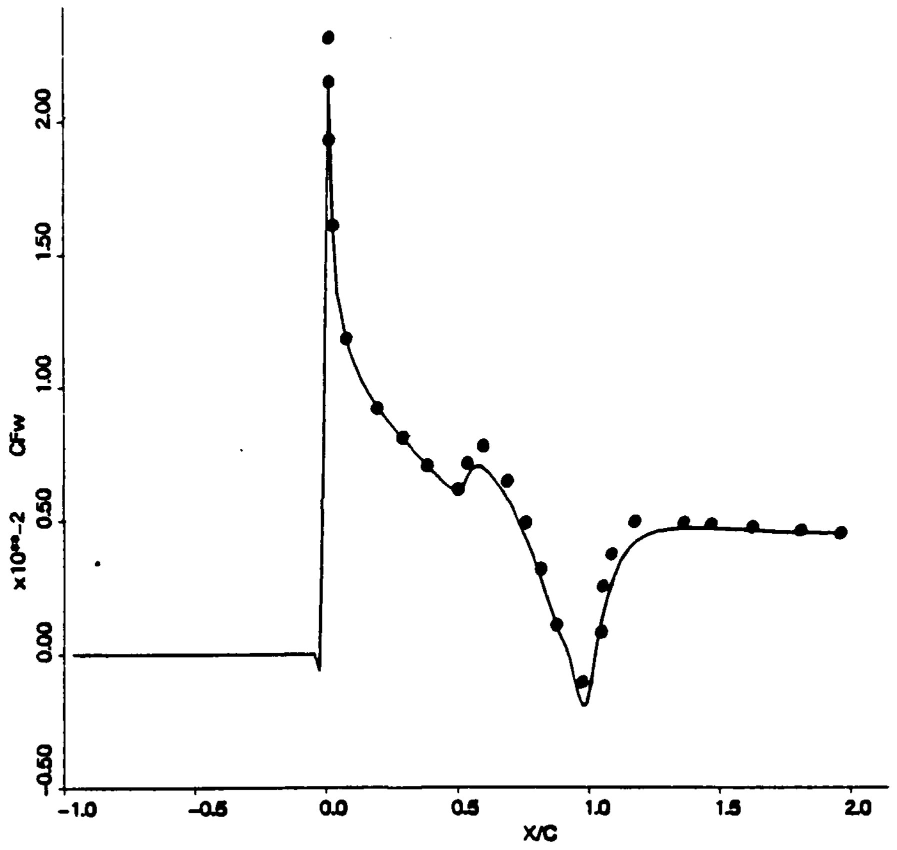
$M_\infty = 0.8 + 0.04 \sin t$
0.95
0.90
0.85
U
0.80
0.75
0.70
0.65
– globally fine
● embedded
0.
1.
2.
4.
5.
6.
Periods


station 1
station 2

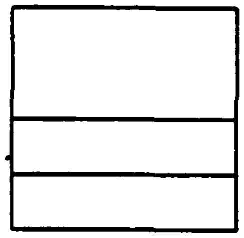

viscous/inviscid boundary
station 1
station 2

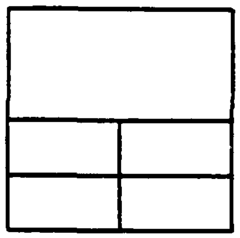

A_2
A_3
Level 0
0
A_1
A_4
Level 1

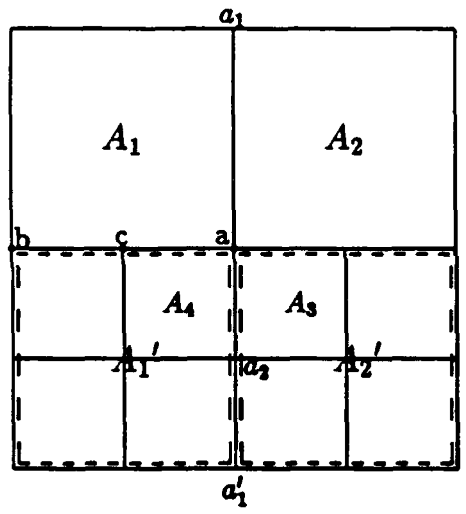


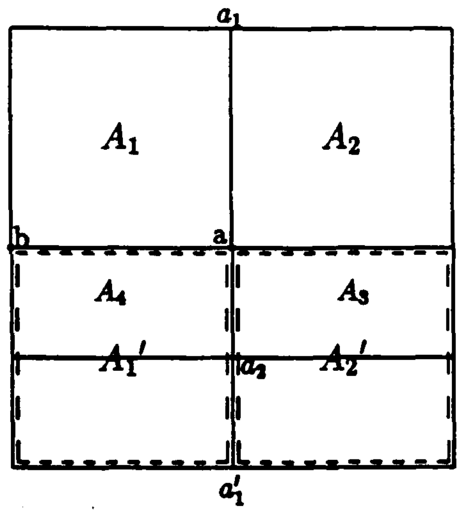


x10**-2 CFw
2.00
1.50
1.00
0.50
0.00
-0.50
-1.0
-0.5
0.0
0.5
1.0
1.5
2.0
X/C


a

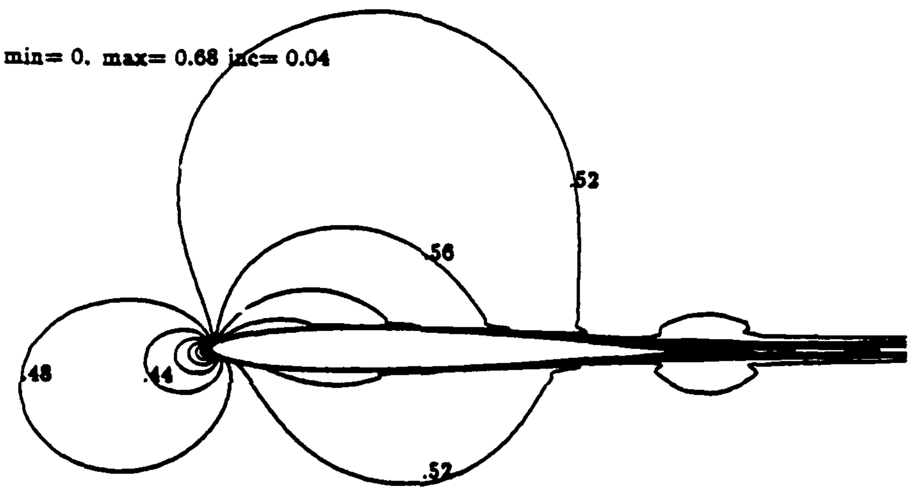

b

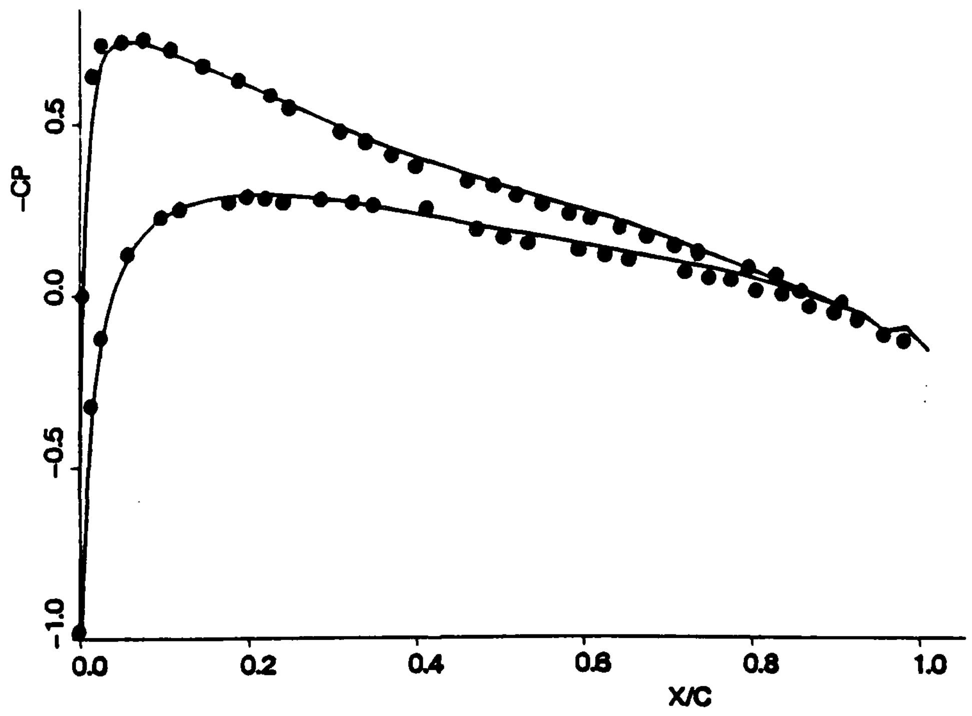

a_1
A_1
A_2
b
c
a
A_4
A_3
A_1'
a_2
A_2'
a_1'


a_1
A_1
A_2
b
a
A_4
A_3
A_1'
a_2
A_2'
a_1'

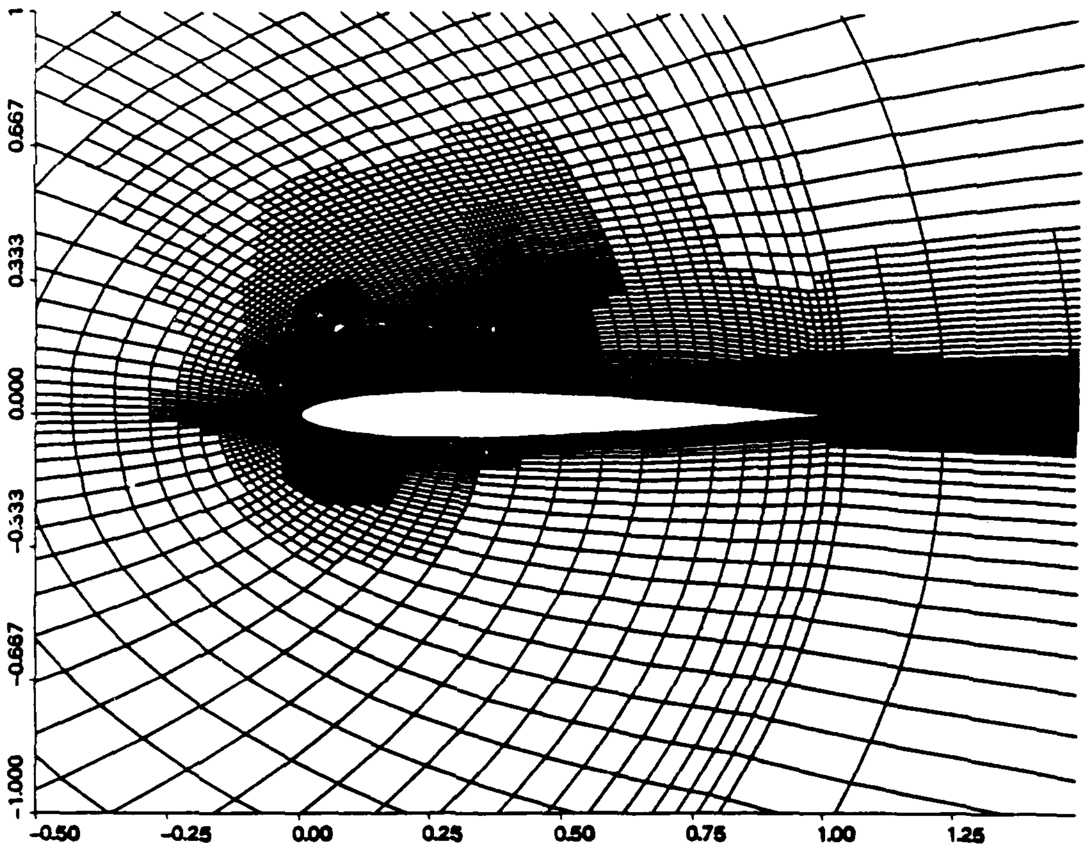
Mach Contours

min= 0. max= 0.68 inc= 0.04



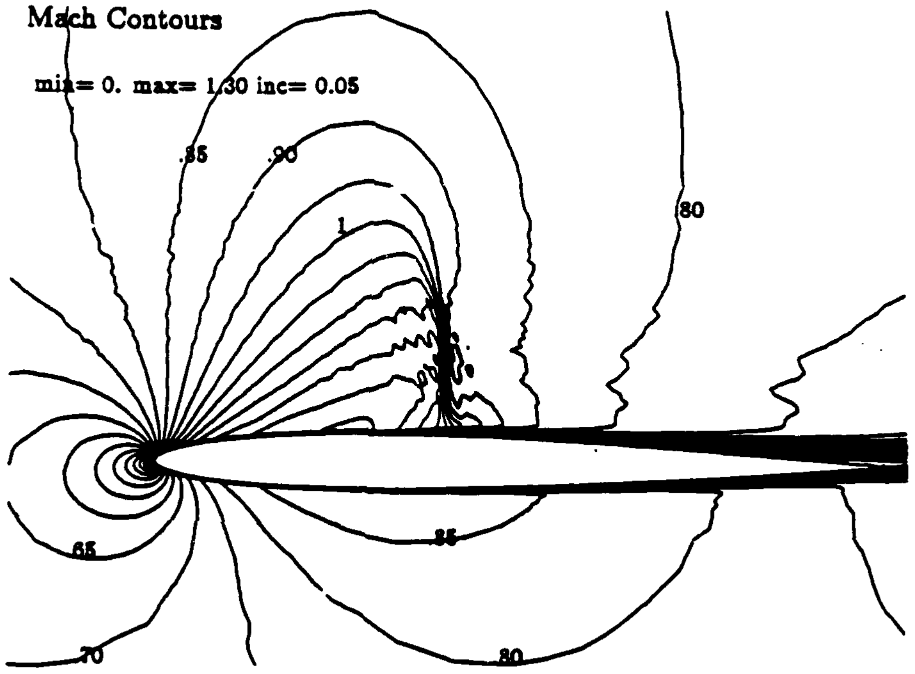


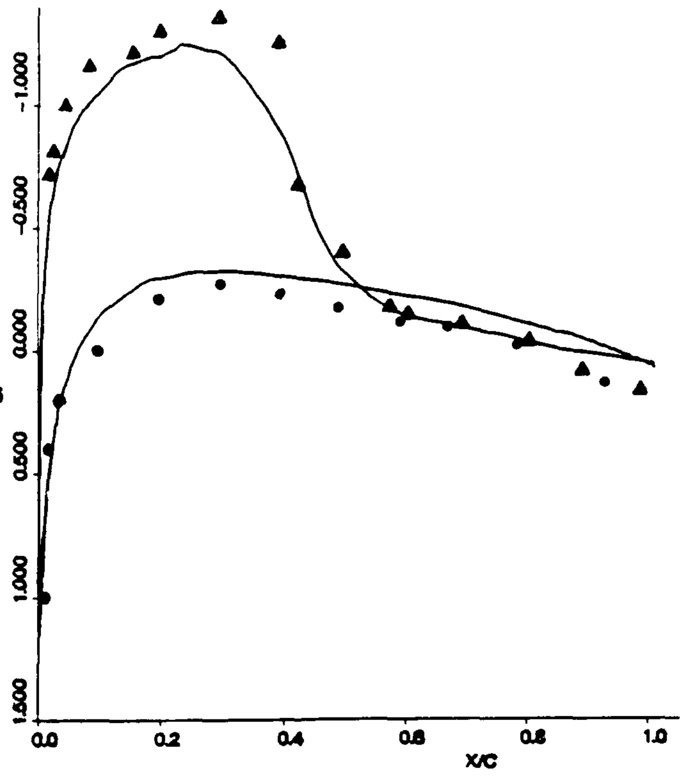
-CP
0.5
0.0
-0.5
-1.0
0.0
0.2
0.4
0.6
0.8
1.0
X/C


1
0.667
0.333
0.000
-0.333
-0.667
-1.000
-0.50
-0.25
0.00
0.25
0.50
0.75
1.00
1.25

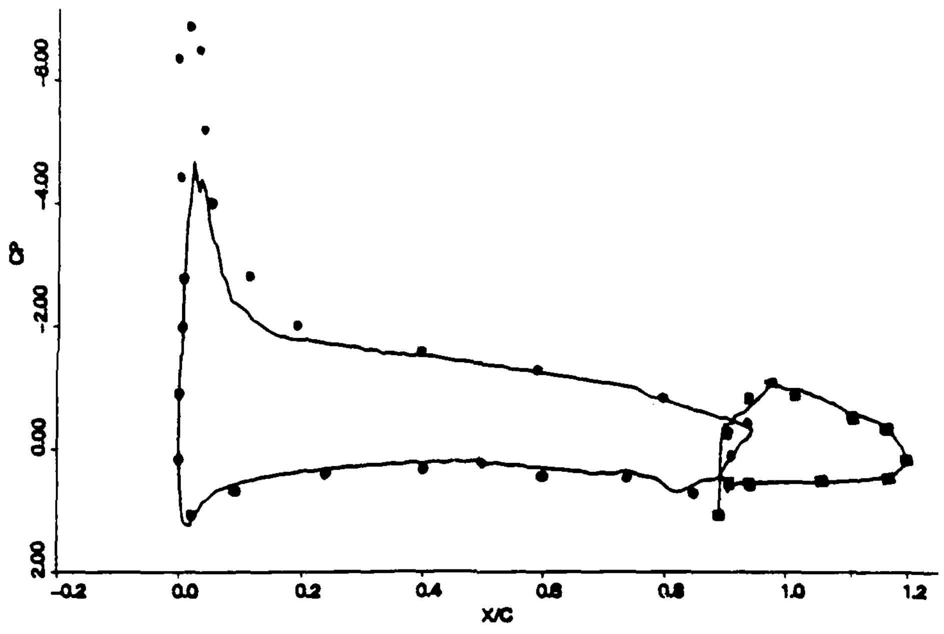
Mach Contours

min= 0. max= 1.30 inc= 0.05

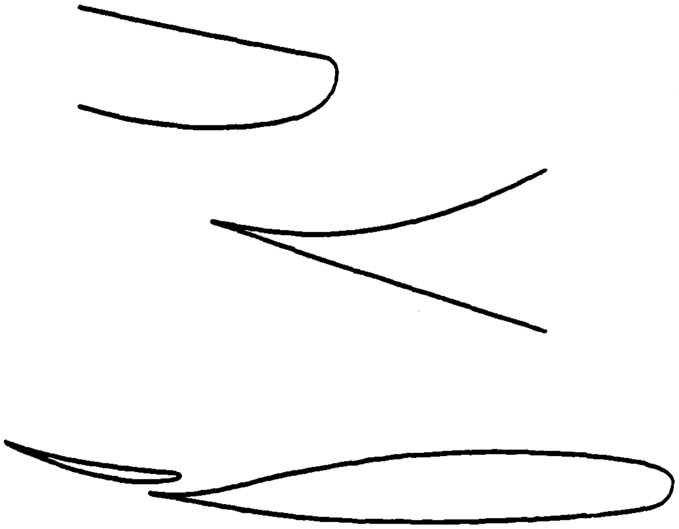


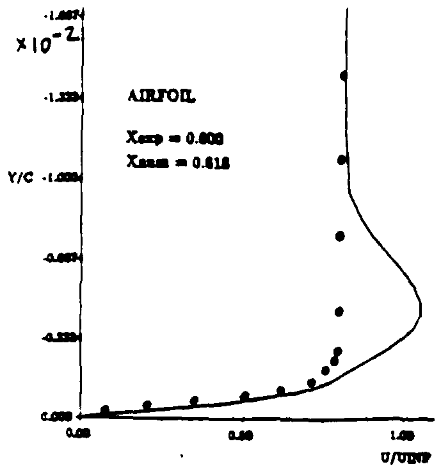


-1.000
-0.500
0.000
0.500
1.000
1.500
Cp
0.0
0.2
0.4
0.6
0.8
1.0
X/C

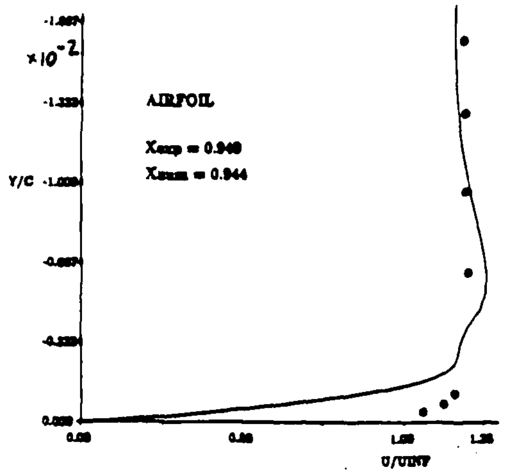


CP
-6.00
-4.00
-2.00
0.00
2.00
-0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
X/C

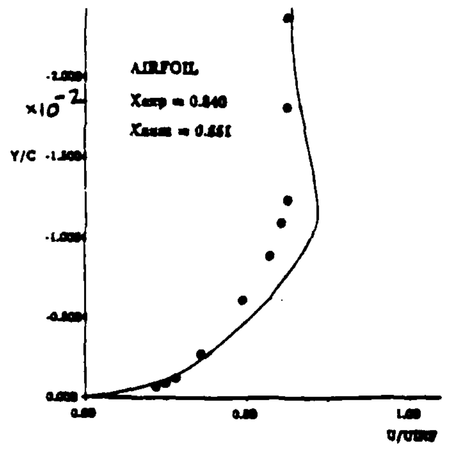




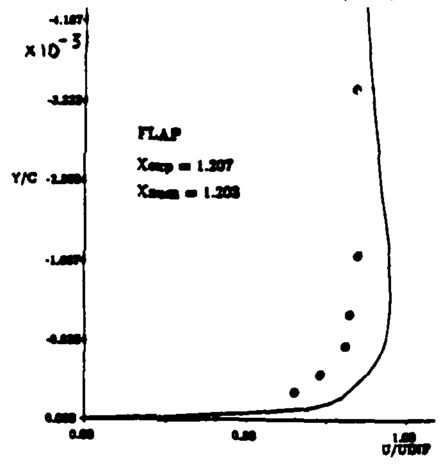
$\times 10^{-2}$
AIRFOIL
Xexp = 0.800
Xnum = 0.818
Y/C
-1.667
-1.333
-1.000
-0.667
-0.333
0.000
0.00
0.50
1.00
U/UINF



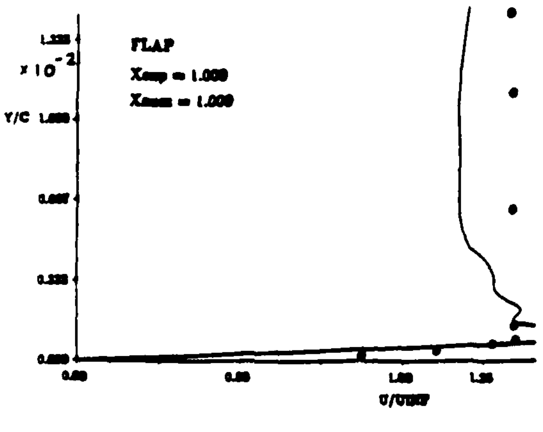
$\times 10^{-2}$
AIRFOIL
Xexp = 0.940
Xnum = 0.944
Y/C
-1.667
-1.333
-1.000
-0.667
-0.333
0.000
0.00
0.50
1.00
1.25
U/UINF



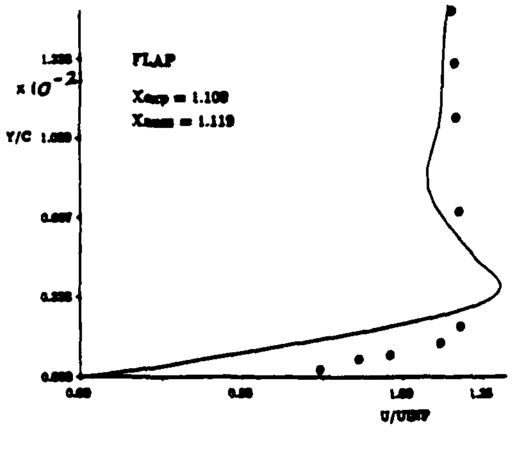
$\times 10^{-2}$
AIRFOIL
Xexp = 0.840
Xnum = 0.851
Y/C
-2.000
-1.500
-1.000
-0.500
0.000
0.00
0.50
1.00
U/UINF



$\times 10^{-3}$
FLAP
Xexp = 1.207
Xnum = 1.208
Y/C
-4.167
-3.333
-2.500
-1.667
-0.833
0.000
0.00
0.50
1.00
U/UINF



$\times 10^{-2}$
FLAP
Xexp = 1.000
Xnum = 1.000
Y/C
1.333
1.000
0.667
0.333
0.000
0.00
0.50
1.00
1.25
U/UINF



$\times 10^{-2}$
FLAP
Xexp = 1.100
Xnum = 1.119
Y/C
1.333
1.000
0.667
0.333
0.000
0.00
0.50
1.00
1.25
U/UINF